L161—O-1096

UIUCDCS-R-76-771

AN EXPERIMENT ON CAI SEQUENCING CONSTRUCTS

By

DAVID W. EMBLEY

February, 1976

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

UIUCDCS-R-76-771

AN EXPERIMENT ON CAI SEQUENCING CONSTRUCTS

By

DAVID W. EMBLEY

February, 1976

Department of Computer Science
Univeristy of Illinois at Urbana-Champaign
Urbana, Illinois 61801

This work was supported in part by the National Science Foundation under Grant No. US-NSF-EC-41511.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This report describes and gives the results of an experiment designed to investigate sequencing constructs in computer-aided instruction (CAI) programs. The experiment compared two languages. One incorporated sequencing constructs of an established CAI programming language; the other contained alternate constructs which stressed uniformity, orthogonality, and static definition and minimized the use of hidden side effects.

Working on-line, subjects debugged and modified a large program while the system monitored their progress and gathered data. Although several interesting results emerged, it was impossible to state conclusively that the entire set of sequencing constructs in either language proved better. Instead, the experiment indicated that direct, explicit, and simple constructs are best. Recommendations are given for designing sequencing features for CAI languages.

ACKNOWLEDGMENTS

Professor Wilfred J. Hansen deserves a special thanks for guiding me through this study. He not only discussed the experiment with me on numerous occasions, but he also allowed me to conduct the experiment as part of his CS 317 course on computer-aided instruction.

I thank Professor Richard G. Montanelli for his help on the statistical aspects of this study. He also read an early draft of the manuscript and offered several pertinent and important suggestions.

Many others helped in several wasy. The CS 317 students cooperated beyond expectations. Ron Danielson suggested some improvements for the experiment. Professor George H. Friedman gave technical and administrative assistance, and Kathy Gee typed this report.

1. INTRODUCTION

1.1 Psychological Studies in Programming

Programming languages have traditionally been designed by individuals or relatively small groups. Language designers generally introduced and propagated language features based on their own feelings, thoughts, and experiences. They rarely consulted with the intended user community and made little or no effort to determine the psychological soundness of newly designed language constructs. One objective method to consult the wishes of the user community and concurrently establish psychological soundness of new language constructs is through thorough, carefully documented, and replicable experiments [1]. Through empirical tests, designers can gain assurance that their language features actually meet the needs of the intended user community, and they can present evidence to support their claims about stylistic and language design issues.

In 1971, Gerald M. Weinberg wrote The Psychology of Computer Programming [2] to trigger the study of computer programming as a human activity. Although most of the book concentrates on social and individual activity in programming, Weinberg also hoped to encourage language designers to include the psychology of computer programming as a new dimension in their design philosophy.

Since the appearance of Weinberg's book, an increased number of researchers have conducted psychological experiments on programming language features. Using inexperienced programmers, three psychologists concluded that nested *if* programs are easier to understand than corresponding *goto* programs [3]. Larry Weisman listed and catagorized factors affecting the complexity of programs, began development of a methodology for studying the psychological complexity of computer programs, and conducted some experiments attempting to determine which factors or combinations of factors reduce complexity [4,5]. At the

University of Indiana, memorization tests were used to measure the effect of program structure on understanding [6]. John D. Gannon, in his Ph.D. thesis, gathered empriical evidence to support or discredit specific language design decisions [7]. Recently, the author reported the results of an experiment which investigated the psychological soundness of a new control construct called the KAIL selector [8].

This report describes yet another experiment intended to experimentally verify hypotheses about programming language design issues. It explores alternatives for expressing sequencing constructs in computer-aided instruction programs.

1.2 The KAIL Project

This work is part of the Automated Computer Science Education System (ACSES) project [9] which uses the PLATO CAI system [10] to teach elementary computer science. At this writing, PLATO runs on a CDC Cyber 73 dual processor with two million 60-bit words of extended core storage. Attached to the processor are over nine hundred plasma display terminals, which were invented by the PLATO group for this project. The peak comfortable system load is about four hundred simultaneous users.

Under PLATO, all programs are written in a special system language called TUTOR [11] which mixes FORTRAN-like flow of control with peculiar CAI sequencing and sophisticated answer judging facilities. The language KAIL is an attempt to influence the development of TUTOR by showing that modern control constructs and sequencing features can be combined with the higher level features of TUTOR.

1.3 General Complaints about TUTOR

Peculiar sequencing rules (see section 2.2) make TUTOR code difficult to read; moreover, they are often a source of bugs. In order to understand

a listing, an author must memorize the rules of unit, jump, help, inhibit, etc. Bugs arise because authors forget, for instance, that goto automatically returns to the end of the current main unit. Often, students become lost in lessons either because unanticipated function keys are active or normal PLATO defaults are inactive. These errors can generally be attributed to an author's misunderstanding of sequencing rules.

In opposition to TUTOR, a CAI language which follows more generally accepted sequencing rules should encourage authors to write more reliable code. The experiment described in this report tests this general hypothesis.

2. THE LANGUAGES AND HYPOTHESES

2.1 The Two Languages of the Experiment

The languages designed for the experiment emphasize differences in sequencing constructs and minimize other differences as much as possible. The common features include the KAIL selector [12, 13] which was previously tested for psychological soundness [8], input-output statements similar to those found in TUTOR, and declarations and expressions typical of high-level programming languages. One language, T, incorporates TUTOR sequencing, and the other language, K, contains an alternate specification for sequencing, similar to that proposed for KAIL. The important differences between K and T are discussed in the sections below, and the pertinent pages of the reference manuals are contained in appendix A.

2.2 TUTOR and the T Language

TUTOR's approach to computer-aided instruction, in its simplest form, is based on a sequence of "display-response" frames. A TUTOR program repeatedly displays information on the screen and then accepts and processes student responses. A lesson author codes a particular "display-response" frame as a unit and strings units together to build a lesson using a variety of inter-unit connections. The language T is based on TUTOR.

2.2.1 Program Structure

In T, a unit is coded as a procedure. A T program or lesson consists of a sequence of procedures optionally preceeded by global variable declarations.

<T-program> ::= lesson <identifier>;
[<variable declarations>;]
<procedures>;
end

<procedures> ::= <procedure>|<procedure>; <procedures>

<procedure> ::= <procedure heading>; <statements>; end

Procedure nesting and local declarations are not permitted. Execution begins with the first statement in the first procedure and flows into other procedures via internally and externally initiated transfers of control.

The system initiates an internal transfer of control to another procedure when it encounters one of the branch statements, jump, goto, or procedure call.

<branch statements> ::= jump <procedure name>
|goto <procedure name>
|<procedure name>

If control enters a procedure via jump, entry into a lesson, or an externally initiated transfer of control, it is a main procedure; and if control enters a procedure via goto or procedure call, it is an attached procedure. Attached procedures always return control to a procedure in the calling routine. For procedure calls, control returns to the statement textually following the call; for goto, control returns to the end of the main procedure.

2.2.2 Exception Handling

Externally initiated transfers of control are called exceptions. A student presses a function key to raise an exception.

<function key> ::= next|back|help|lab|data
|<shifted function key>

<shifted function key> ::= nextl|backl|helpl|labl|datal

To process exceptions, the system maintains an internal pointer for each function key. This pointer is active or enabled, if the pointer references

a specific procedure in the lesson; it is inactive or disabled if the pointer is null. The system transfers control to the designated procedure when a student raises an exception that has an enabled pointer.

When control enters a main procedure, all pointers are set to null except the next pointer which is set by default to the textually next procedure. When execution reaches the end of a main procedure, the system automatically pauses to wait for student input; and if the student presses next, control passes to the procedure designated by the next pointer. By executing an exception enabling statement, a lesson author can explicitly set any pointer, including the next pointer.

<exception enabling statement>::= <function key name> <procedure name>

For example,

help hint;

helpl big-hint;

causes the help pointer to reference procedure hint and the helpl pointer to reference procedure big-hint.

The additional semantics of exception handling in T depend on the function key name in an exception enabling statement. Next, nextl, back, and backl evoke lesson sequences, and help, helpl, lab, labl, data, and datal evoke help sequences. Lesson sequences are completely independent of any execution history; help sequences are not.

When a student raises an exception which evokes a help sequence, the system designates the current main procedure as the base procedure and sets an internal pointer to it. Following the help sequence, control returns to the beginning of the base procedure. As shown in figure 1, a help sequence consists of one or more main procedures. In a help sequence, the system not only enables the next pointer as usual, it enables the back and backl pointers as well and associates them with the base procedure. When execution encounters

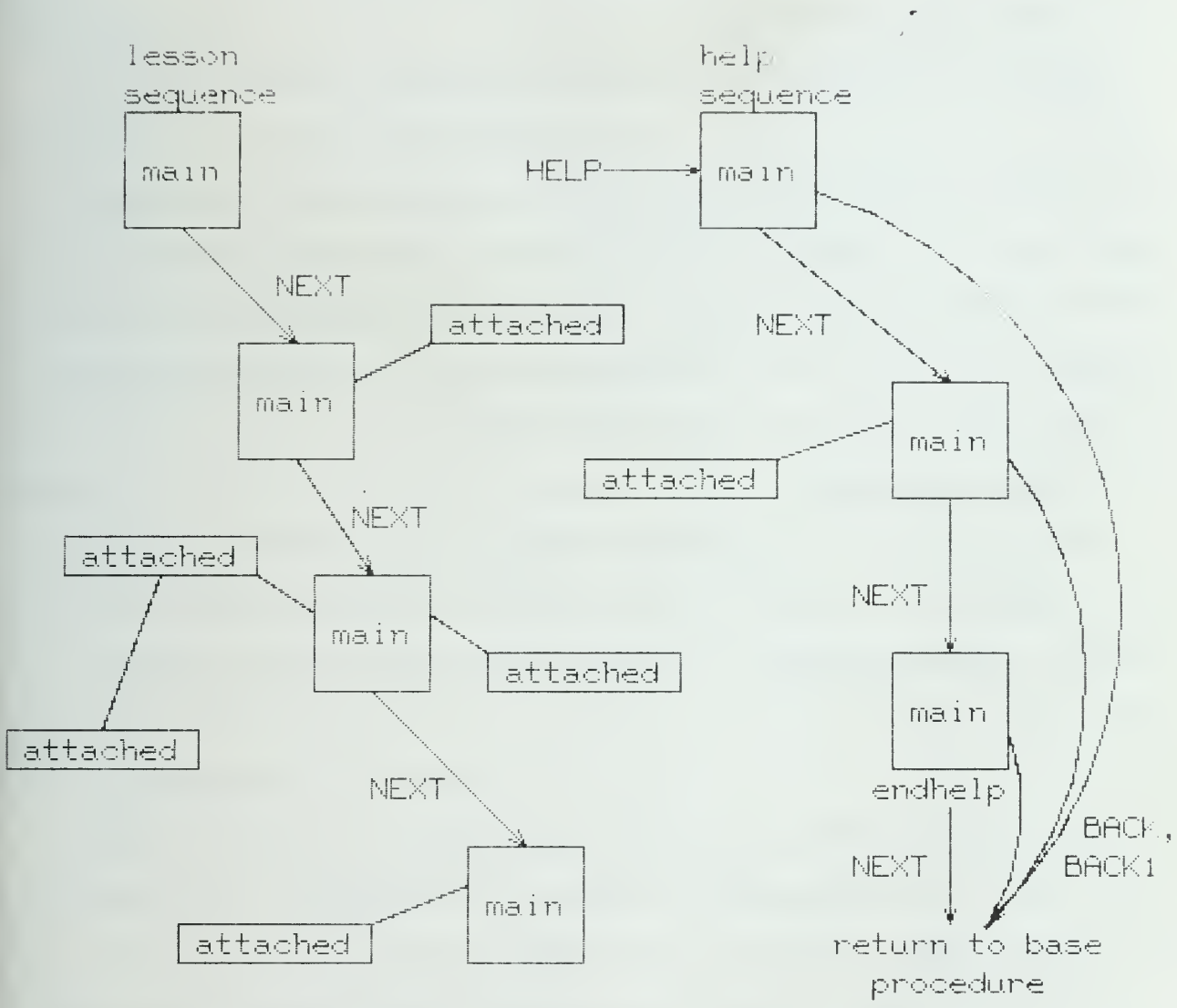


Figure 1. Lesson and Help Sequences in T. If the help sequence is evoked from one of the main or attached procedures in the lesson sequence, the main procedure in the calling routine becomes the base procedure to which control returns following the help sequence.

an endhelp in place of a simple end, the next pointer is set to the base procedure, and the help sequence terminates.

2.2.3 Modifications

T modifications statements are defined as

<modification statement> ::= rewrite|echo|erasure|size<expression>
|rotate <expression>| long <expression>

These commands modify input-output statements. For the write statement, rewrite, echo, and erasure set the mode, and size and rotate designate the character size and angle of writing respectively. The long statement limits the length of an input string. Modifications are set and reset dynamically, and their effect extends beyond procedure boundaries.

2.2.4 Hidden Side Effects

When the execution of a statement causes an action unrelated to the main function of the statement or not denoted by the statement name, this action is a hidden side effect. For example the T jump statement causes a screen erase. Hidden side effects permeate the T sequencing features. For each main procedure, the system automatically generates a screen erase at the beginning and a pause at the end. If a lesson author wishes to prevent these defaults, an inhibit erase statement nullifies automatic screen erase; and a jump statement transfers control without pausing.

In lesson sequences, the system automatically associates next with the textually next procedure. In help sequences, it automatically associates back backl with the base procedure, and activates next as in lesson sequences except on encountering endhelp.

Other features such as the returning goto can also be considered as hidden side effects.

2.3 Hypotheses

There are at least four problems with T sequencing constructs. Each has led to a hypothesis tested in this experiment.

2.3.1 Uniformity

T lacks uniformity. One example is the nonuniform behavior of procedures. Syntactically, procedures appear identical, but the semantics depend on whether a procedure connects into a sequence via jump, goto, subroutine call, textual sequential ordering, or student raised exceptions. A common error is to insert an attached procedure between two main procedures and then evoke it by falling through from the textually preceeding main procedure. When languages lack uniformity, many rules and exceptions to rules are needed. This causes infrequently used constructs to be forgotten, makes deviation from the basic norm difficult, and discourages casual users from gaining in depth understanding. UNIFORMITY HYPOTHESIS: programs would be easier to understand and use if written in a language where syntactically similar constructs have similar semantics.

2.3.2 Orthogonality

T lacks orthogonality. When language facilities are highly independent, they are said to be orthogonal. An example of interdependence in T is the tight binding of the semantics of lesson and help sequences to particular function keys. This binding discourages authors from using exceptions in other contexts where they might be useful. For example, in T it is difficult to code an on-page help sequence, a help sequence where help text appears on the same page as instructional text and where control resumes at the point of interruption rather than the beginning of the base procedure. One solution to this problem might be to introduce a special exception handling statement with its own rules and semantics, but this only compounds the problem. Features which are interdependent and tightly bound together lack generality and

thus, create a need for further extensions, rules, and exceptions to rules. ORTHOGONALITY HYPOTHESIS: an orthogonal language would be easier to understand, modify, and use.

2.3.3 Dynamic Execution

T depends extensively on dynamic execution. T exception enabling statements are activated dynamically. This allows an author, for instance, to enable the help key in an attached procedure far removed from the calling routine where the exception may be raised. It also allows an author to sprinkle several specifications for a single function key throughout a procedure and activate them depending on the flow of control. T modifications also depend on dynamic execution. A common mistake is to set size and then forget to reset it. Dyanmic execution features are error-prone because they impose fewer restrictions on the author. DYNAMIC EXECUTION HYPOTHESIS: by properly limiting the dynamic nature of these and similar statements, it is likely that the resulting language would be less error-prone.

2.3.4 Hidden Side Effects

T sequencing constructs have many hidden side effects. Each of these is useful and makes coding easier provided an author understands them all and uses them correctly. T authors commonly forget to inhibit unwanted side effects and often use explicit statements where basic defaults are already provided. Also, different needs may create situations which oppose basic defaults. These needs usually force authors to write awkward sections of code. Sometimes further extensions (e.g. on-page help) might alleviate these awkward situations, but the extensions often impose new defaults or additional side effects. The number of rules and exceptions to rules soon grow beyond an author's ability to remember. HIDDEN SIDE EFFECTS HYPOTHESIS: although hidden side effects usually reduce the amount of coding, it is likely that by eliminating most of them, a language would be easier to understand and use.

2.4 The K Language

K was designed to propose alternate solutions to T's sequencing constructs. It stresses uniformity, orthogonality, and static definition and minimizes the use of hidden side effects.

2.4.1 Program Structure

A K program is defined as

<K-program> ::= lesson <identifier>; <constructs>; end

<constructs>::= <construct>|<construct>; <constructs>

<construct> ::= <statement>
|<variable declaration>
|<procedure declaration>
|<exception declaration>

Procedures contain local variable and exception declarations but not local procedure declarations. Execution begins with the first statement and ends with the last statement. Only explicit calls evoke procedures; thus, K treats all procedures uniformly.

2.4.2 Exception Handling

In K, exception handling is defined as

<exception declaration> ::= on <function key> [<exception handler>]

<exception handler> ::= <statement>

K enables exceptions statically and thus, avoids dynamic execution as a means of activation. If an exception is raised in a block containing an associated exception declaration, control immediately passes to its handler. After executing the handler, control continues with the statement textually following the handler unless a goto explicitly transfers control elsewhere. The principle of orthogonality is observed because the function key name in an exception declaration and the handler are totally independent.

To keep an exception active in a called procedure, the exception must be passed as a parameter. A parameter list in K is defined as

```
<parameter list> ::= <parameter set>|<parameter set> ; <parameter list>
<parameter set>  ::= <type>[ result ] <identifier list>
                    |on <function key list>
<type>           ::= integer|real|string (<integer>)
```

When the system detects an exception in a procedure associated with an exception declaration in its parameter list, control immediately returns, the system raises the exception in the calling routine.

2.4.3 Modifications

K modifications are defined as

```
<modifications>      ::= {<modification list>}
<modification list>  ::= <modification>|<modification> , <modifications>
<modification>       ::= rewrite|echo|erasure|size <expression>
                        |rotate <expression>| long <expression>
```

K modifications avoid dynamic execution problems because a modification only affects the statement to which it is attached. Thus,

```
{size 0.75} [ at 429;  {size 3}write TITLE;
              at 605;  write sentence;]
```

writes "TITLE" in size 3, but writes "sentence" in size 0.75. When a statement to which modifications are attached includes a procedure call, the modifications carry into the procedure. For example, the following code places a box on the screen and then erases it.

```
drawbox(x,y);   {erasure}drawbox(x,y);

                    .
                    .
                    .

procedure drawbox(integer x,y);

    draw x,y..x+100,y..x+100,y+100..x,y+100..x,y;

    end;
```

2.4.4 Hidden Side Effects

K contains no hidden side effects. The language requires every action to be explicit.

3. THE EXPERIMENT

3.1 Overview

To test the hypotheses stated in chapter 2, it was necessary to devise a method to measure a subject's ability to understand and use a language. Several factors were involved.

Personal factors:

1. Educational experience
2. Knowledge of programming language
3. Basic intelligence
4. Motivation

Observable factors:

1. Accuracy in coding
2. Speed at finding and correcting program bugs
3. Facility at finding and correcting bugs and at making modifications

Together, these factors formed a model of subject understanding. This model of a subject's ability to perform programming tasks broke the hypotheses down into tests involving measureable variables.

For the experiment, subjects were divided into two groups, the T language group and the K language group. The effects of personal factors were controlled for by grouping subjects so that background and motivation in both groups were similar. After receiving instruction in their assigned languages, subjects were asked to debug and modify a reasonably large lesson consisting of about 500 lines of code. Subjects performed this task on-line while the system gathered data.

3.2 Subjects

The experiment took place during the Fall Semester, 1975, and the subjects were students in CS 317. Professor W. Hansen, instructor, required each student to participate in the experiment. As part of the course, the students learned TUTOR and used it to code a substantial CAI lesson. In order to expose them to the additional language concepts needed for the experiment, the course included a section on KAIL. Each student read through lesson KAIDS [13], an introduction to KAIL, coded the control flow for a typical KAIL lesson, and took a quiz to test their comprehension.

Based on this quiz and a background information sheet, subjects were divided into two groups. To accomplish this division, each subject received a score in the indicated range on the following items:

quiz score (0 - 37),

status at the university (FR=1,...,G1=5, G2=6)

number of CS courses taken (0 - 6),

understanding of

ALGOL 68 (1 - 5)

KAIL (1 - 5),

TUTOR (1 - 5),

PLATO experience (1 - 10).

Subjects were ranked according to the sum of these scores. (Those with identical sums were ranked alphabetically.) Using this ranking, subjects received a group assignment according to a predetermined random grouping schema. The quiz, background information sheet, and subject grouping schema are contained in appendix B.

3.3 Lectures on T and K

Once divided into groups, subjects learned their assigned language in a one-hour lecture given by the author. Since they were already familiar with

TUTOR and KAIL, it was only necessary to review the applicable language features for each group and emphasize important points. Because of scheduling problems, subjects attended these lectures in small subgroups ranging in size from one to six. Neither language group, however, gained an advantage by having more small subgroups. At these lectures subjects signed up for their session on PLATO; these were scheduled between 5 and 8 days, after their lecture.

3.4 The PLATO Session

The on-line portion of the experiment proceeded as depicted in Figure 2.

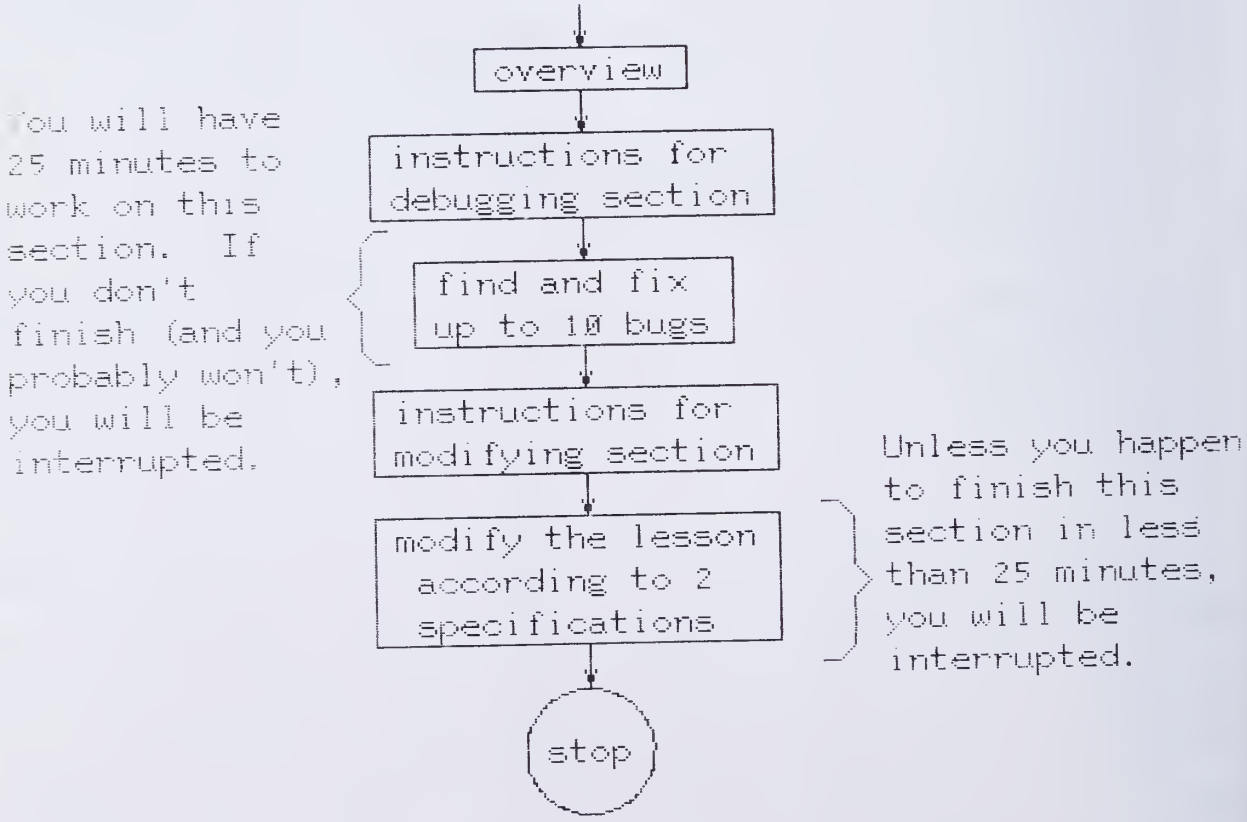


Figure 2. Overview of the Experiment

Each subject was asked to debug and modify lesson "intrep" on integer representations written in the subject's assigned language, K or T. The logical structure for "intrep" was as depicted in Figure 3. The solid arrows indicated paths through the lesson; the broken arrows indicated paths to the quiz. A student taking "intrep" was to complete every section before attempting the quiz (i.e., before traversing a broken path).

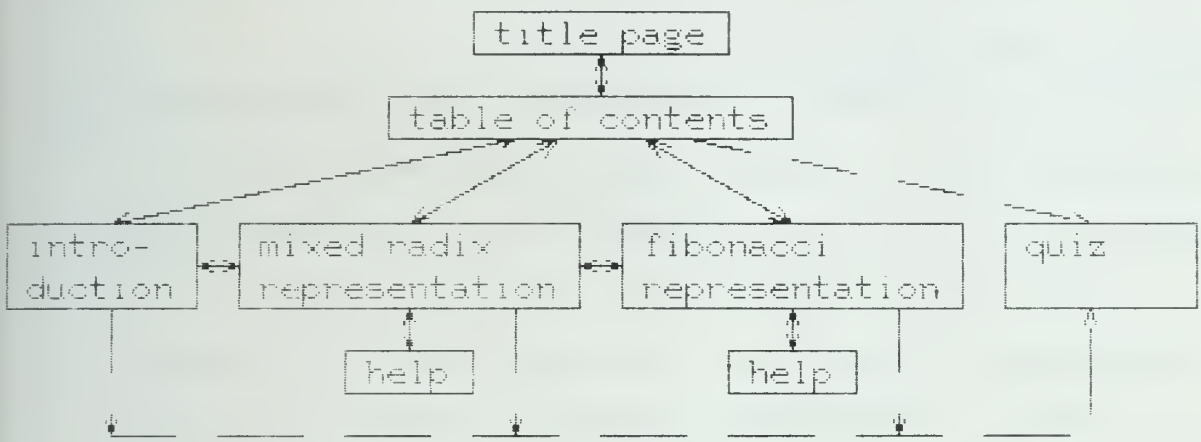


Figure 3. Flow Diagram for "intrep". Solid lines indicate paths through the lesson; broken lines indicate paths to the quiz.

Several on-line aids were available during the experiment:

(1) A complete program listing. The listings of "intrep" for the the two languages appear in Appendix C.

(2) An editor with the following commands:

Fn to move forward n lines

Bn to move backward n lines

In to insert text after line n

Rn to replace line n

Dn to delete n lines

Xt to locate text t

(3) A reference manual. The essential pages of the reference manuals for K and T are in Appendix A. These contained syntax diagrams and a minimal explanation of the semantics.

(4) Program execution. A subject could execute "intrep" to assist in locating and fixing program bugs.

(5) The flow diagram for "intrep", figure 3.

In the debugging section, subjects had 25 minutes to find as many of the bugs as possible. The system seeded the program with bugs one at a time and in a specific order. Thus, at any given moment, the program contained only one bug. Their task was to find it and fix it.

All bugs were logic errors and typified those which might actually arise in programming "intrep". Naturally, bugs were selected which related to the hypotheses, but care was taken to keep them typical and thus, hopefully, fair.

To help subjects identify a bug, the system directed them through a path in the execution of the lesson leading to the problem and then gave a one line explanation. To shorten the path leading to an error, the system began each path at a convenient point, not necessarily the title page. Subjects were able to execute "intrep" as desired to help locate the bug.

When a subject pressed the term key and entered "editor" the program listing became available for editing. While editing, subjects could return again to the execution of "intrep" with the bug. Any changes to the code, however, were not reflected in the execution of the program.

After subjects fixed a bug (or gave up), the system presented a solution to the problem. The subject's solution was graded later by hand.

As an example, figures 4, 5, and 6 show one bug and its solution, first in K and then in T.

BUG: screen overwritten. (TERM editor)

TABLE OF CONTENTS

An everyday, ordinary, base B integer N is represented as

a. Introduction

$$N = (b_n b_{n-1} \cdots b_2 b_1 b_0)_B$$

$$= b_n B^n + b_{n-1} B^{n-1} + \cdots + b_2 B^2 + b_1 B + b_0$$

Are there other ways to represent N?

d. Quiz

Press a, b, c, or NEXT.

Figure 4. BUG: Screen Overwritten

```
21.3 minutes remaining   OUR SOLUTION
 1  procedure intro;
 2       on back;  on back1 goto indx;
 3       at 321;   { size 1.5 }write A. INTRODUCTION:
 4       at 905;  write
 5
 6  ~     An everyday, ordinary, base B integer N is represented
 7  as
 8
 9            N = (b_n b_n-1 ... b_2 b_1 b_0)_B
10
11              = b_n B^n + b_n-1 B^n-1 + ... + b_2 B^2 + b_1 B + b_0
12
13
14       Are there other ways to represent N?;
15
16       at 1915;  accept;
17       at 1915;  erase 2+length(reply);  at 1710;  erase 36;
18       at 1705;  write
19
20  ~     This is only one of many representations for N.  In
21  this lesson, (nrsctns-2) other systems are discussed:
22
23                          1. Mixed Radix
24
```

After line 2, insert

```
erase;
```

Press NEXT

Figure 5. Solution in K

```
2.5 minutes remaining   OUR SOLUTION
 1  procedure index;
 2       back title;
 3       term "index","contents";
 4       erase;
 5       at 320;  size 1.5;  write TABLE OF CONTENTS;  size 0;
 6       loc ← 1117;
 7       tnr ← 1;
 8       *[ while | tnr≤nrsctns :
 9            [ if | tcompl(tnr) :  at loc-2;  write *; ];
10            topics;
11            loc ← loc+300;
12            tnr ← tnr+1;
13       ]*;
14       loc ← [ if qok | 3018 | 3020 ];
15       at loc;  write Press a, b, c, ;
16       [ if | qok :  write d, ; ];
17       write or NEXT.;
18       whereto_[ pause;  if key
19            | "a" :  goto intro;
20            | next :  goto intro;
21            | "b" :  goto mixedrad;
22            | "c" :  goto fibonac;
23            | "d" :
24                 [ if qok | jump quiz; | at 3105;  write
```

Replace the goto's in lines 19 through 22 with jump's.

Press NEXT

Figure 6. Solution in T

After completing the debugging section, subjects were given an additional 25 minutes to modify a bug-free listing of "intrep" according to the following specifications:

Specification #1:

Add a help sequence so that whenever HELP is pressed from the table of contents page, a page intended to describe the key conventions used throughout the lesson appears. Code the necessary flow of control, but just have the system display "key conventions" on the page where the explanation would appear. When NEXT, BACK, or SHIFT BACK is pressed at the end of the help sequence, the student is to be returned to the table of contents page.

Specification #2:

Add a help sequence so that whenever HELP is pressed during the quiz, a line of explanation appears on the same page saying, "Sorry, no help available during the quiz." This line of explanation is to be erased on the first press of NEXT, BACK, or SHIFT BACK, and the student is to be returned to the point of interruption.

While editing, subjects could refer to these specifications as often as desired. The system imposed no constraints on how subjects were to accomplish the task; thus, they could modify the program as they wished. Their efforts were graded later by hand.

4. RESULTS

4.1 Measures of Ability

As indicated in section 3.1, a subject's ability to comprehend and perform was measured in several ways:

(1) accuracy in coding:
    (a) subjective scores
    (b) number of coding errors
    (c) type of coding errors

(2) speed at finding and correcting each program bug:
    (a) time spent executing "intrep"
    (b) time spent searching the listing
    (c) time spent altering the listing
    (d) time spent using the aids
    (e) total time

(3) facility at finding and correcting each bug and at making modifications:
    (a) number of times "intrep" was executed for the debugging section, and number of times the specifications were accessed for the modifying section
    (b) number of times editor searching commands were used
    (c) number of changes of the listing
    (d) number of times aids were referenced
    (e) total number of editing keypresses

These performance statistics appear in appendix D, E, and F, and the important results are discussed below.

4.2 Difficulties

Several difficulties were encountered during the experiment.

Two K subjects and one T subject dropped out of the experiment. This left an imbalance of 13 K subjects and 15T subjects. T-tests showed, however, that this did not bias the experiment.

During one session, the system crashed. Because this problem was anticipated, experiment sessions on PLATO were limited to at most four subjects, two from each group. The crash caused the loss of partial data from two T language subjects and one K language subject.

In other sessions, three subjects accidentally terminated the experiment prematurely. A T language subject accidently hit stopl, a built-in escape feature from any PLATO lesson. Misunderstanding the directions in the modification section caused two K language subjects to prematurely press the combination of keys provided for exiting the experiment. In all of these cases, partial data was lost.

In analyzing the results, an unexpected number of language interference errors appeared. K subjects used a few TUTOR-like constructs, and T subjects used a few KAIL-like constructs. Despite the interference, it was relatively easy to decide what caused a subject to choose a particular syntactic form or expect some specific semantic action.

4.3 Background

As discussed in section 3.2 subjects were grouped so that background and ability would be equal. An analysis of the statistics confirmed this; there were a few specific areas, however, in which unexpected differences occurred. A typical subject in the K language group was a first year graduate student, while a typical subject in the T language group was nearly,

but not quite, a senior at the university. This was almost close enough to b be considered significant ($p \leq .059$). More K subjects had taken CS 323, system programming ($p \leq .024$); and K subjects better understood ALGOL-W ($p \leq .014$). Both groups had similar PLATO experience and KAIL quiz scores. Table 1 summarizes the background and ability data.

Personal Data	Means		
	K	T	T
university status	4.38	3.60	1.97
course taken	3.38	2.40	1.62
language understanding	27.38	25.40	1.18
PLATO experience	4.31	4.40	.19
quiz score	28.92	29.47	.39

Table 1. Summary of Background and Ability Data

4.4 Debugging Statistics

The debugging section of the experiment was completed by 12 of 13 K subjects and 12 of 15 T subjects. The other 4 encountered the difficulties discussed in section 4.2. Table 2 shows the extent to which subjects from each group worked the problems. T subjects worked more problems (5.8 on the average) than K subjects (4.8 on the average), but this is not statistically significant. Bugs 1 through 5 are discussed below; not enough subjects worked problems 6 through 10 to allow statistical analysis.

4.4.1 Bug 1

The first bug tested the dynamic execution hypothesis. A title in "intrep" was supposed to be written in size 1.5, but the statement

```
write C. FIBONACCI REPRESENTATION;
```

wrote it in size 0. K language subjects should have fixed this bug with

```
{size 1.5} write C. FIBONACCI REPRESENTATION;
```

and T language subjects should have fixed it with

```
size 1.5; write C. FIBONACCI REPRESENTATION;  size 0;
```

Problem Number	Group	Completed Problem	Worked on Problem	Only Looked at Problem
1	K	11	0	2
	T	14	0	1
2	K	13	0	0
	T	14	0	0
3	K	10	2	0
	T	12	0	2
4	K	8	2	0
	T	11	2	0
5	K	5	0	3
	T	8	0	3
6	K	2	2	0
	T	5	2	0
7	K	1	0	0
	T	3	0	0
8	K	0	1	0
	T	0	2	0
9	K	0	0	0
	T	0	0	0
10	K	0	0	0
	T	0	0	0

Table 2. The Extent to which Subjects Worked the Problems. There were 13 K subjects and 15 T subjects.

Only 1 of 14 T subjects who attempted the problem solved it correctly, whereas 5 of 11 K subjects solved it correctly (table 3). An exact [14] showed this was significant ($p \leq .039$). Moreover, only 3 subjects in the T group made any attempt to reset size, and another 3 had the failure to reset size as their only error. The 11 K subjects made 8 errors; the 14 T subjects made 18 errors. Timing and editing data showed essentially no differences.

	number of errors		
	0	1	≥ 2
K	5	4	2
T	1	9	4

Table 3. Error Contigency Table for Bug 1

This data supports the dynamic execution hypothesis. Subjects committed fewer coding errors when using static modifications.

4.4.2 Bug 2

The second bug tested the hidden side effects hypothesis. In the K listing, an unwanted erase was placed at the end of a help sequence; in the T listing, a necessary inhibit erase was deleted. Of 13 K subjects, 8 performed well; of 14 T subjects, 8 also performed well. Two T subjects, however, placed inhibit erase in the wrong procedure; apparently, they understood the problem but not the rules of inhibit erase. The K group introduced 0.23 errors per person; the T group introduced 0.31 errors per person. Although K subjects spent less time altering the listing ($p \leq .009$), this only indicates that inserting takes more time than deleting. No other significant differences emerged.

Apparently, it was no more difficult to detect the absence of an inhibit erase and insert it than to find and delete an unwanted erase. This bug was probably too simple to reveal any significant information about the hidden side effects hypothesis.

4.4.3 Bug 3

The third bug considered program structure and exception handling and tested hypotheses relating to uniformity and dynamic execution. Coding CAI lesson sequences with traditional constructs seemed to demand a level of indirection

(goto label 1 ... 1: call procedure p), but T sequencing (continue at procedure p) did not. Figure 7 shows K's indirect method of going back to the previous frame; figure 8 shows T's direct method. The bug was seeded into the programs by deleting

on back goto ttl;

from K's listing and

back title;

from T's listing.

Only 1 of 10 K subjects who completed the problem solved it correctly, whereas 7 of 12 T subjects solved it correctly (table 4). An exact test [14] showed this was significant ($p \leq .026$). Of the 10 K subjects, 8 entered goto <procedure name>; moreover, 2 T subjects used KAIL constructs but also entered goto <procedure name>. The 10 K subjects made 16 errors; the 12 T subjects made only 9 errors. On the average, the T subjects scored 8.1 (10 possible), and the K subjects scored 5.6 ($p \leq .013$). Moreover, even though the amount of code to be inserted for a correct solution was nearly identical for both languages. K subjects spent more time inserting ($p \leq .030$), perhaps because they were unsure of themselves.

	number of errors		
	0	1	≥ 2
K	1	3	6
T	7	5	0

Table 4. Error Contingency Table for Bug 3

Contrary to the general hypothesis (section 1.3), traditional sequencing in K resulted in poorer performance. The K language subjects seemed confused by going one place to get somewhere else.

```
                                .
                                .
                                .
          ttl.      title;
          indx.     index;
                                .
                                .
                                .
     procedure title;
          .
          .
          .
          end title;
     procedure index;
          on back goto ttl;
          .
          .
          .
          end index;
```

Figure 7. Handling back in K

```
.
.
.
procedure title;
     .
     .
     .
     end title;
procedure index;
     back title;
     .
     .
     .
     end index;
```

Figure 8. Handling back in T

On the other hand, 2 T subjects placed "back title" in the wrong procedure. In both instances, these subjects may have expected back to remain active once it was enabled since under this assumption, they essentially solved the proble correctly.

This lends support to both the uniformity and dynamic execution hypothesis. The 2 T subjects seemed to be confused about the interaction between dynamic activation of exceptions and entrance into a main procedure.

4.4.4 Bug 4

The fourth bug tested the hidden side effects hypothesis. In T, the jump statement, in addition to directing control to a designated procedure, erased the screen. The goto only directed control elsewhere and could often be used in place of a jump. Bug 4 was a screen overwrite problem. An erase was left out of the K listing; a goto, instead of a jump, was used in the T listing.

In T, 10 of 11 subjects gave a correct solution, but only 1 replaced the goto with a jump. The rest, 9 of 10, gave an alternate solution: they properly inserted erase in the appropriate procedure. Clearly, T subjects found it easier to fix the bug directly. In K, 5 of 8 solved the problem correctly. Both groups fixed the bug in the same manner; however, K subjects inserted more often ($p \leq .045$), activated more searching options ($p \leq .015$), and pressed more editing keys ($p \leq .040$). There is no clear explanation why K subjects edited more than T subjects.

The results lend support to the hidden side effects hypothesis because the explicit solution was preferred. Underlying implicit effects were more difficult to find.

4.4.5 Bug 5

The fifth bug tested the hidden side effects hypothesis. Because help sequences in T returned to the beginning of the base procedure, procedures had to be broken at unusual junctures; and because the screen erased automatically when entering a main procedure, normal defaults had to be overridden. Solving both these problems at once resulted in particularly awkward and unusual code. Figure 9 shows the essential statements in a section of T code where unusual coding was necessary. The inhibit erase prevented an unwanted screen erase on entering the help sequence, mrhelp. The erase in mixedr2 was essential because the automatic erase which normally would have occurred in falling through from mixedr1 was inhibited. Figure 10 shows the corresponding section of K code. In both instances, the bug was created by removing the erase.

Of the 8 T subjects who attempted to solve this problem, 3 removed inhibit erase from procedure mixedr1 which fixed the immediate problem but introduced another bug. All 5 K subjects who attempted this problem solved it correctly. K subjects scored 14.0 of 15 in a subjective measure which

adjusted the score to give more credit to those who did well quickly, but T subjects scored only 9.6. This is significant ($p \leq .049$).

This supports the hidden side effects hypothesis and shows how combinations of hidden side effects interfere with one another ot produce unexpected bugs.

```
.
.
.
procedure mixedrad;
.
.
.
    end mixedrad;
procedure mixedrl;
    .
    .
    .
    inhibit erase;
    help mrhelp;
    .
    .
    .
    at 3020;  write Press NEXT to continue.;
    end mixedrl;
procedure mixedr2;
    erase;
    .
    .
    .
    end mixedr2;
```

Figure 9. Section of T Code

```
.
.
.
procedure mixedrad;
    .
    .
    .
on help mrhelp;
    .
    .
    .
    at 3020;  write Press NEXT to continue.;
    pause next;
    erase;
    .
    .
    .
    end mixedrad;
```

Figure 10. Section of K Code

4.5 Modifying Statistics

The modifying section of the experiment was completed by 10 of 13 K subjects and 13 of 15 T subjects. The other 5 encountered the difficulties discussed in section 4.2. Refer to section 3.4 for the spcifications used in the modifying section of the experiment.

K subjects made several errors related to the hypotheses of the experiment. Most of these related to the hidden side effects hypothesis and were probably a result of TUTOR language interference. On 11 occasions of a possible 20, no pause occurred after help messages. Subjects may have been counting on a nonexistent automatic pause at the end of a procedure. K subjects also seemed

confused about erasing after help messages: in 5 places in connection with the first specification, an unnecessary erase appeared; and in 4 places in connection with the second specification, the message was left on the screen. Contrary to the general hypothesis (section 1.3), K subjects continued to include errors of the from

goto <procedure>

A total of 10 of these errors crept into the listings. This further supports the results found in bug 3 and shows that this kind of indirect transfer to a frame processor is probably unnatural.

T subjects also made several errors related to the hypotheses of the experiment. With regard to the uniformity hypothesis, T subjects positioned 5 of a possible 26 attached procedures in sequences of main procedures so that the attached procedures would also be executed as main procedures. In the second specification, no one solved the pause problem correctly. With the expection of those who did even worse, everyone included a double pause in approximately the form shown in figure 11. One pause occurred explicitly; the other occurred implicitly at the end of the help sequence. This relates to the hidden side effects hypothesis, but it also relates to the orhtogonality hypothesis because the implicit pause at the end of the help sequence is bound to rather than separate from the semantics of the help sequence. Other errors were related to the hidden side effects hypothesis. On 9 of 26 occasions, subjects did not take advantage of normal defaults and referred unnecessarily to next, back, and backl. T subjects omitted endhelp in 5 of 26 help sequences and forgot to include inhibit erase in 6 of 13 instances where it was required.

```
.
.
.
procedure sorry
     inhibit erase;
     at 3010; write sorry;
     pause next,back,back1;
     at 3010;  erase 5;
     endhelp sorry;
```

Figure 11. Double pause Problem

All subjects in both groups coded the control for the second problem improperly. The specification called for control to resume at the point of interruption in the quiz. Most subjects, however, wrote code which simply restarted the quiz; and no one even attempted to force control to resume at the interruption point. The problem was difficult enough that subjects were unable or unwilling to solve it.

In general, observations on the modification section of the program support the hidden side effects hypothesis. In 25 minutes, 10 K subjects made made 20 hidden side effects related errors and 13 T subjects made 36 such errors. The K errors seemed to be based on assumptions learned from TUTOR.

4.6 Correlations

Appendix F contains the correlation data for the major factors in the experiment. Using score as a measure of performance, table 5 shows how it correlated with other factors. The number of years of university training had no correlation with performance. Other background data correlated highly with scores from the K language group, but not so highly with scores from the T language group. Programmers with greater insight and ability performed better

in K, but insight and ability did not seem to matter so much in T. Perhaps this was because T, being like TUTOR, was more familiar to all subjects, whereas K, contianing less familiar concepts, allowed experienced programmers to perform better and hindered less experienced programmers even more.

	K Language Group			T Language Group		
	number in group	debugging score	modifying score	number in group	debugging score	modifying score
University status	13	.000	.078	15	.313	.178
Courses	13	.542**	.548*	15	.131	.105
Languages	13	.735**	.779**	15	.379	.210
PLATO experience	13	.785**	.644**	15	.239	.507*
KAIL quiz score	13	.717**	.601**	15	.442	.152
Rank	13	-.901**	-.871**	15	-.567*	-.309
Time, executing "intrep"	12	-.500*	--	12	.012	--
Time, looking at listing (debugging)	12	.482*	--	12	-.438	--
Time, altering listing (debugging)	12	-.381	--	12	-.037	--
Total editing keypresses (debugging)	12	-.587*	--	12	-.541*	--
Time, reading specifications	10	--	-.412	13	--	-.402
Time, looking at listing (modifying)	10	--	.293	13	--	-.269
Time, altering listing (modifying)	10	--	-.410	13	--	.024
Total editing keypresses (modifying)	10	--	.018	13	--	-.487*

* significant ($p \leq .05$)

** significant ($p \leq .01$)

Table 5. Data Correlated with Scores

5. CONCLUSIONS

5.1 Summary of Results

Although some significant and interesting results emerged, it was not possible to state conclusively that the entire set of K sequencing constructs was better than the set of T sequencing constructs. Instead, the results were mixed.

The experiment did show, however, that direct, explicit, and simple constructs were best. Indirect frame sequencing in K, implicit constructs in T, and the lack of a simple resume feature in both K and T seemed to cause most of the errors. In both bug 3 and the modifying section, indirect sequencing patterns confused K subjects and resulted in poorer performance. This was the only significant contradiction to the general hypothesis that more generally accepted sequencing rules would encourage authors to write more reliable code. Misunderstanding hidden side effects and interference with TUTOR hidden side effects hindered T and K subjects respectively. Both groups committed several coding errors which seemed to be based on expectations about implicit actions such as automatic pause or erase. Because T lacked an on-page help feature, and because K lacked a resume feature, there was no simple solution to the control problem for the second specification in the modifying section of the experiment. Because of this difficulty, subjects were unwilling or unable to force control to resume at the point of interruption.

5.1.1 Unfiormity

The data from the experiment gave support to the uniformity hypothesis. In the modifying section, T subjects introduced errors by improperly inserting attached procedures into the listing. K's uniform treatment of procedures prevented K subjects from making this mistake.

5.1.2 Orthogonality

The data from the experiment supported the orthogonality hypothesis. In the modifying section, all T subjects found the double pause problem particularly difficult to avoid. In K, procedure and pause were independent, so the problem did not occur.

5.1.3 Dynamic Execution

The experiment confirmed one instance of the dynamic execution hypothesis. When dealing with dynamic modifications in bug 1, T subjects introduced more coding errors than K subjects. Less dynamic K modifications were easier to understand and use.

5.1.4 Hidden Side Effects

The results supported the hidden side effects hypothesis. In bug 4, T subjects preferred to fix a screen overwrite problem directly by inserting an erase rather than indirectly by changing a goto to a jump obtaining erase as a side effect. In bug 5, the inhibit erase/erase phenomenon caused misunderstandings for some T subject.

As exhibited by errors which emerged from the modifying section of the experiment, T subjects misunderstood aspects of hidden side effects and K subjects allowed the TUTOR knowledge to interfere. Some K subjects left out necessary pauses and seemed confused about erasing after help messages. Many T subjects referred unnecessarily to next, back, and back1 and forgot to include a needed inhibit erase.

5.2 Recommendations

These recommendations for CAI language features resulted from the data of the experiment.

1. Allow direct jumps to frame processors. Because CAI is heavily based on meaningful sequence of display-response frames, an author language should allow programmers to directly and simply code

frame processing and sequencing. The experiment demonstrated that this deficiency in K resulted in poor performance. The data did not imply, however, that all subroutines should be coded as frame processors as in T.

2. Limit dynamic execution of modifications. Although more tests would be necessary to show that the syntactic form

{<modifications>}<statement>

is best, the results indicate that this semantic restriction should be placed on modification.

3. Eliminate most, if not all, hidden side effects. As expected, hidden side effects adversely affected performance. More testing would be necessary, however, to firmly state that all hidden side effects should be eliminated or that a specific few should be retained.

4. Include a resume feature as part of exception handling. As exemplified in the modifying section of the experiment, the lack of this facility would discourage authors from writing code to permit a student to obtain help and then resume at the point of interruption. This need occurs commonly in CAI.

5.3 Suggested Improvements

Although performance measures on accuracy in coding and observations on the specific kinds of coding errors produced a reasonable number of interesting results, statistics on the speed and facility at fixing program bugs and on the facility at making modifications were disappointing. It was hoped that the time spent executing "intrep", searching the listing, and the total time spent locating and fixing a bug would yield information about which constructs make debugging faster and thus easier, and it was hoped that the total number of editing keypresses would indicate which

constructs and program structure make a programmer work harder to debug and modify a lesson. The experiment would have been improved if subjects could have observed the result of their changes when executing "intrep". This would have forced them to reconsider their fixes and modifications in light of whether or not they actually solved the problem. It is likely that more obscure constructs would have caused subjects to spend more time and do more work to solve the problems. As it was, however, the system accepted any solution and left it unchallenged.

The number of interference errors was larger than expected, and more K subjects committed interference errors than T subjects. This indicates that the experiment would have been improved by spending more time teaching the assigned languages and by insuring a greater similarity in a subject's TUTOR and KAIL background.

Three other changes might have improved the experiment.

1. The correlation between university status and performance indicates that this factor should not have been considered as part of a subject's background data.
2. Although 25 minutes was about the right amount of time for the modifying section of the experiment, subjects needed more than 25 minutes for the debugging section.
3. The choice of bugs for this kind of experiment is always very difficult. The experiment might have been imporved by a different choice of bugs, but the chosen set seems to have been reasonable.

REFERENCES

[1] Shneiderman, Ben, "Experimental Testing in Programming Languages, Stylistic Considerations, and Design Techniques," Technical Report No. 16, Indiana University, Computer Science Department, October, 1974

[2] Weinberg, Gerald M., The Psychology of Computer Programming, Van Nostrand Reinhold Company, New York, 1971.

[3] Sime, M. E., T. R. G. Green, and D. J. Guest, "Psychological Evaluation of Two Conditional Constructions Used in Computer Languages," International Journal of Man-Machine Studies, Vol. 5 No. 1, 1973, pp 105-113.

[4] Weissman, Laurence M., "Psychological Complexity of Computer Programs: An Experimental Methodology," SIGPLAN Notices, Vol. 9, No. 6, June, 1974, pp 25-36.

[5] Weissman, Laurence M., "A Methodology for Studying the Psychological Complexity of Computer Programs," Technical Report CSRG-37, University of Toronto, August, 1974.

[6] Shneiderman, Ben, and Mao-Hsian Ho, "Two Experiments in Programming Behavior," Technical Report No. 17, Indiana University, Computer Science Department, October, 1974.

[7] Gannon, John D., "Language Design to Enhance Programming Reliability," Technical Report CSRG-47, University of Toronto, January, 1975.

[8] Embley, David W., "An Experiment on a Unified Control Construct," Technical Report UIUCDCS-R-75-759, University of Illinois, August 1975.

[9] Nievergelt, J., E. M. Reingold, and T. R. Wilcox, "The Automation of Introductory Computer Science Courses," SIGCUI Bulletin, Vol. 8, No. 1, 1974, pp. 15-21.

[10] Alpert, D. and D. L. Bitzer, "Advances in Computer Based Education," Science 167, 1970, pp. 1382-1590.

[11] Sherwood, Bruce A.,"The TUTOR Language", CERL, University of Illinois, June 1974.

[12] Embley, David W. and Wilfred J. Hansen, "The KAIL Selector - A Unified Control Construct," SIGPLAN Notices, Vol. 11, No. 1, January 1976.

[13] Embley, David W., "An Introduction to KAIL," Lessons Kaids on the PLATO System, Univeristy of Illinois, August, 1975.

[14] Lindgren, B. W., Statistical Theory, The MacMillan Company, London, 1968.

APPENDIX A

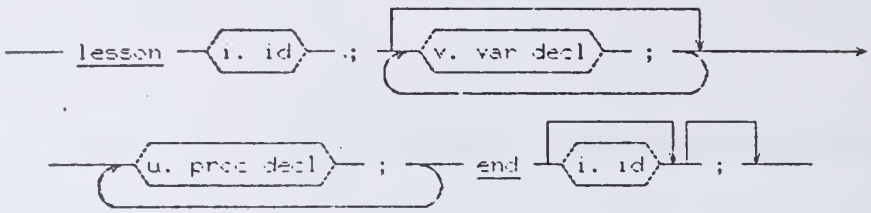
PORTIONS OF THE REFERENCE MANUALS FOR T AND K

Each page in the on-line reference manuals used during the experiment gave a syntax diagram for a particular construct and some explanation of the semantics. A box, indicated that more of the diagram appeared on another page.

The pages of the reference manuals selected for this appendix contain the essential differences between T and K.

A.1 Pages from the T Reference Manual

P. PROGRAM



Execution begins with the first statement in the first procedure and ends after execution of the last statement in the last procedure.

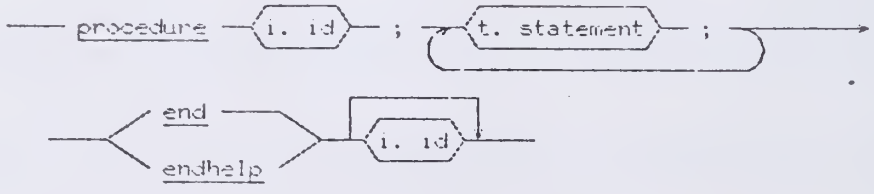
Example:

```
lesson sayhi;

    procedure hi;
        at 1010;
        write 'Hi!';
    end hi;

end sayhi
```

U. PROCEDURE DECLARATION



Procedures behave like TUTOR units.

Use endhelp to terminate a help sequence.

Example:

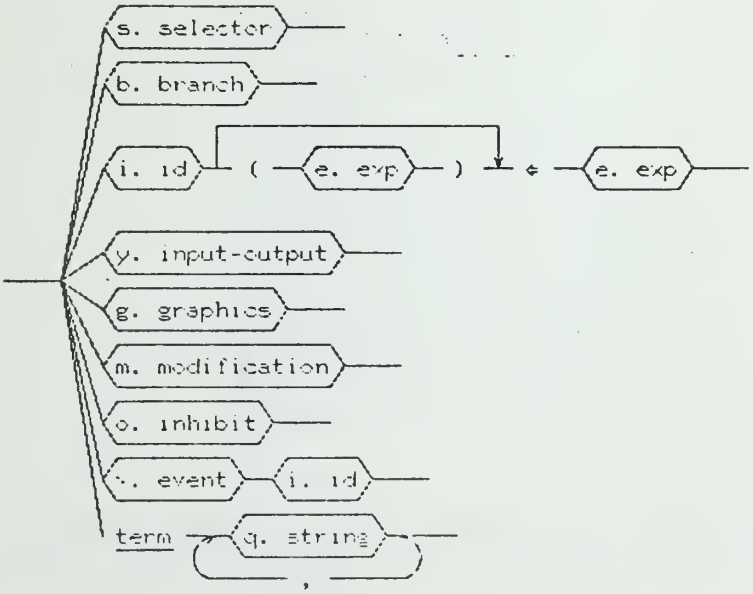
```
procedure wait;
    at 3027;  write 'Press NEXT';
    pause next;
    at 3027;  erase 10;
    end wait;
```

T. STATEMENT

Examples:

```
branch lbl;
lbl. i ← i+1;
size 2.5;  write CONGRATULATIONS;  size 0;
```

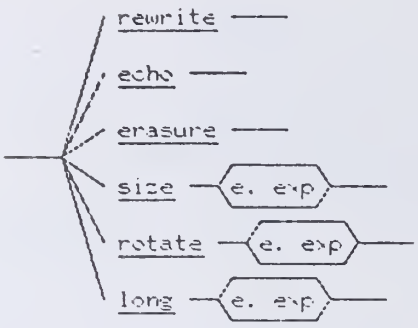
Z. STMT



Examples:

```
i ← i+1
A(i) ← A(i)/denom
V ← (1,2,3,4,5)
term "index","contents"
```

M. MODIFICATION

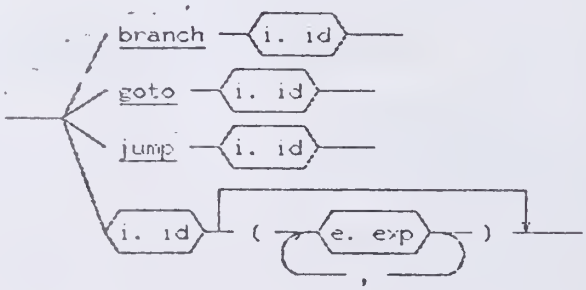


Modifications alter the normal behavior of certain statements. Once set, a modification remains in effect until it is explicitly reset.

Examples:

```
size 2; rotate 45;  write hypotenuse;
    size 0;  rotate 0;
rewrite;  write replace it!;  echo;
```

B. BRANCH

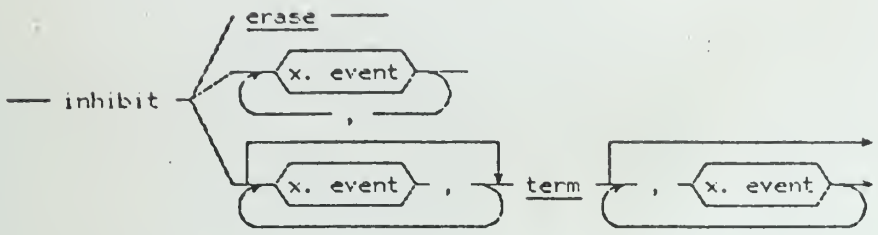


These statements behave like the TUTOR commands to which they correspond. "<id>" corresponds to "do <id>".

Examples:

```
goto index      ¢ "index" is a procedure name ¢
jump axx1       ¢ "axx1" is a procedure name ¢
branch label    ¢ "label" is a label ¢
proc10          ¢ "proc10" is a procedure name ¢
```

0. INHIBIT

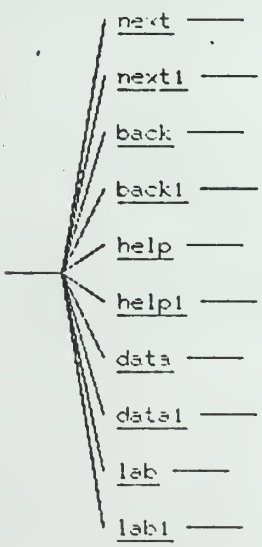


These statements inhibit normal behavior until explicitly altered or until entrance into a new main procedure.

Examples:

```
inhibit data     ¢ makes the data key inactive ¢
inhibit back     ¢ makes the back key inactive ¢
inhibit erase    ¢ inhibits automatic erasures ¢
```

X. EVENT



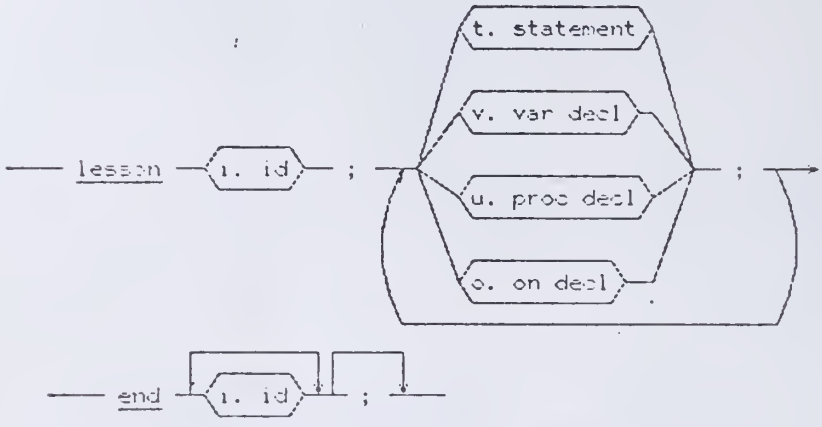
These statements behave like the TUTOR commands to which they correspond.

Examples:

```
next topic3  ¢ sets the next pointer to topic3 ¢
back topic1  ¢ sets the back pointer to topic2 ¢
help hint    ¢ establishes a help sequence, hint ¢
```

A.2 Pages from the K Reference Manual

P. PROGRAM



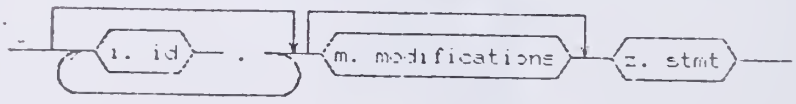
Example:

```
lesson sayhi;

    at 1010;
    write 'Hi';

end sayhi
```

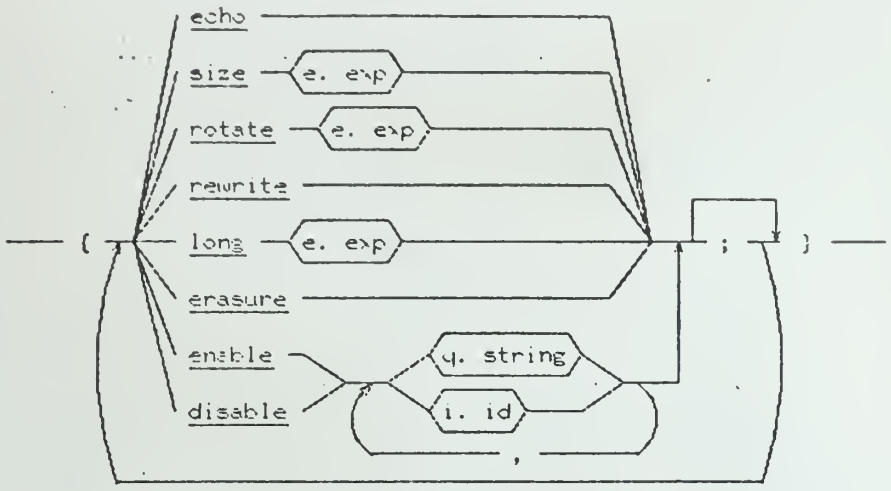
T. STATEMENT



Examples:

```
goto lbl
lbl. i ← i+1
( size 2.5 )write CONGRATULATIONS
```

M. MODIFICATIONS

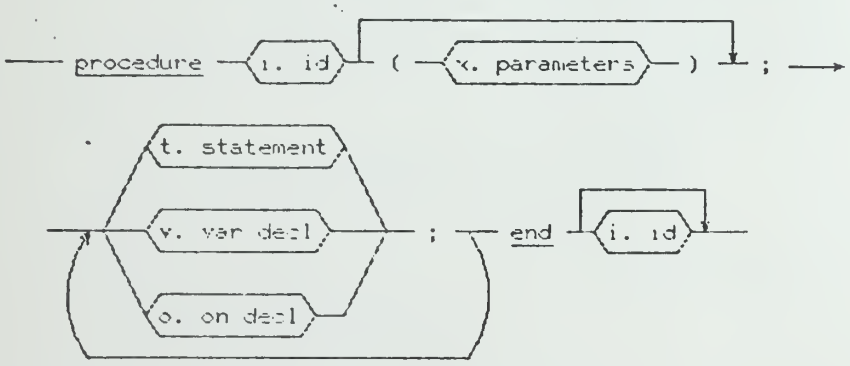


Modifications alter the normal behavior of certain statements to which they are attached. This statement may include a procedure call in which case the modification is in effect while the procedure is active.

Examples:

```
{ size 2; rotate 45 }write hypotenuse
{ rewrite; }write replace it!
{ disable back,back1 } lethargic
```

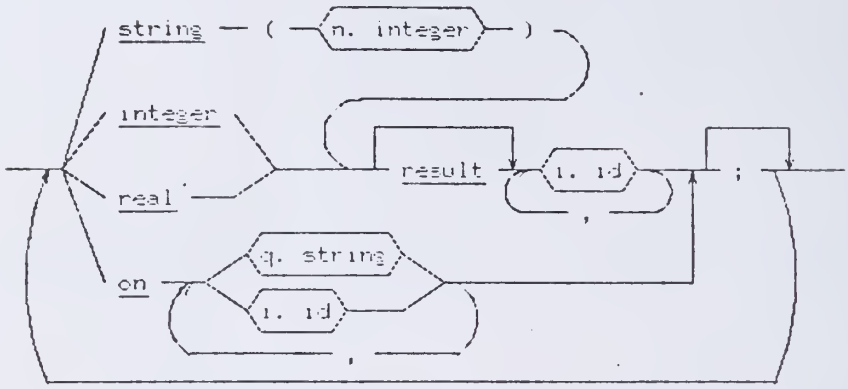
U. PROCEDURE DECLARATION



Example:

```
procedure wait;
    at 3027;  write Press NEXT;
    pause next;
    at 3027;  erase 10;
    end wait;
```

X. PARAMETERS



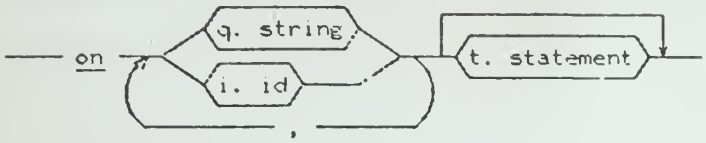
Parameters are called by value unless result is specified.

When events appear as parameters, they are active during the procedure (unless specifically disabled). If they are signaled, the procedure returns immediately and the event is signaled in the calling routine.

Examples:

```
integer min, max;  real x1;
string(10) ud;  on back, "escape"
real result arcat,peecat,qucat
```

0. ON DECLARATION



Like all other declarations, on declarations have static scope.

On declarations, also called event indicators, indicate a course of action to be taken when an event included in the list is signaled. When an event is signaled, the statement associated with the declaration is executed. Afterwards, control passes to the statement following the declaration unless it is explicitly transfered elsewhere.

Several event identifiers are known to the system:

ans	data	help	lab1	stop
back	data1	help1	next	
back1	formerror	lab	next1	

When the user presses one of the function keys, that event is signaled. Formerror is signaled when the system is asked to convert a string to a number and it is not possible.

The term key is active whenever a string literal appears in the on list. When the user presses term and enters an active literal, that event is signaled.

The author may include other identifiers in the event list. These are activated by the signal statement. (See page b.)

Examples:

```
[ on help hint;
    at 1010;  accept;
if reply
    | nearly house :  ok;
    | else :  no;  retry;
];
```

```
i ← 1;
*[ while | ismax :
    [ if names(i)=name :
        signal found;
    ];
    i ← i+1;
]*;
on found nr ← nr+1;
```

APPENDIX B

INFORMATION USED FOR SUBJECT GROUPING

Subjects were ranked based on their background and quiz score and then assigned to a language group according to a predetermined subject grouping schema.

B.1 Background Information Sheet

BACKGROUND INFORMATION SHEET

Name:

Social Security Number:

What is your current status at the University?

1. Freshman
2. Sophomore
3. Junior
4. Senior
5. Graduate student of 1 or 2 years
6. Graduate student of 3 or more years

Indicate which of the following courses (or equivalent) you have taken or are taking.

1. CS 121: Introduction to Computer Programming
2. CS 201: Machine Language and System Programming
3. CS 311: Information Structures
4. CS 323: System Programming
5. CS 325: Introduction to Programming Languages
6. CS 326: Compiler Construction

How well do you understand the following languages?

1. never heard of it
2. know of its existance
3. know a little about it
   or have written a program in it
4. know it fairly well
   or have written several programs in it
5. know practically everything about it
   or have programmed extensively using it

ALGOL 68: 1 2 3 4 5

ALGOLW: 1 2 3 4 5

COBOL: 1 2 3 4 5

EULER: 1 2 3 4 5

FORTRAN: 1 2 3 4 5

KAIL: 1 2 3 4 5

LISP: 1 2 3 4 5

PL/1: 1 2 3 4 5

SNOBOL: 1 2 3 4 5

TUTOR: 1 2 3 4 5

Indicate your PLATO experience.

1. I've never heard of PLATO.
2.
3.
4. I've used PLATO and have done a little authoring.
5.
6.
7. I've written several PLATO lessons.
8.
9.
10. I've been working extensively on PLATO for more than five years.

Note: The numbers which are not labeled are meant to be gradations of experience in between those which are labeled.

B.2 KAIL Quiz

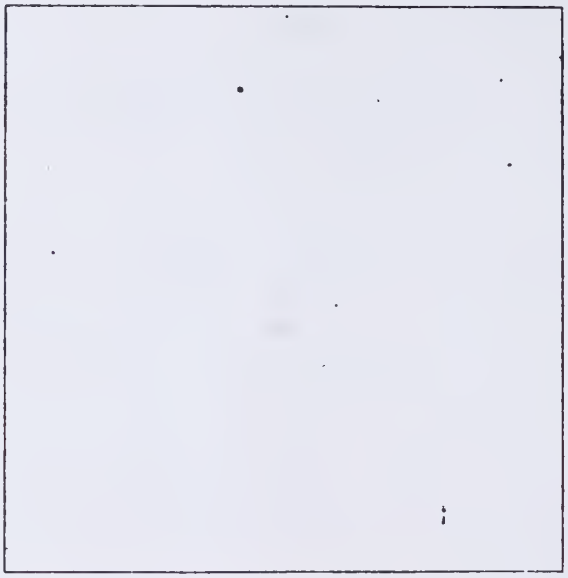
KAIL PROGRAM

```
 1  lesson leapyr;
 2
 3  procedure introduction;
 4      erase;
 5      at 2813;  { size 7;  rotate 45 }write LEAP YEAR;
 6      at 3027;  write Press NEXT;  pause next;
 7      end introduction;
 8
 9  procedure question;
10      integer yr;
11      on back goto intro;
12      erase;
13      at 1010;  write 'What is the first leap year after 1898?';
14      [ on help hint;
15          on formerror clean;
16          at 1215;  { long 6 }accept yr;
17      if yr
18          = 1904 :  write 'Right, very good.;
19          = 1902 :  no;  at 1417;  write
20              A leap year must be divisible by four.;
21              retry;
22          = 1900 :  no;  at 1417;  write
23              When a centesimal year is involved,
24              it must be divisible by 400.;
25              retry;
26          else :  write 'Wrong, try again;  retry;
27      ];
28      at 3027;  write Press NEXT;  pause;
29      end question;
30  procedure hint;
31      at 1817;  write
32          A leap year is every fourth year but only
33          those centesimal years divisible by 400.;
34      pause;  clean;
35      end hint;
36
37  procedure conclusion;
38      on back goto quest;  on back1 goto intro;
39      erase;
40      at 1217;  { size 2.5 }write END OF LESSON;
41      pause next,back,back1;
42      end conclusion;
43
44  intro.  introduction;
45  quest.  question;
46          conclusion;
47
48  end leapyr;
```

QUIZ

NAME: SOCIAL SECURITY NUMBER:

1. In the box, reproduce (approximately) the contents of the screen at the first pause after the program begins execution. (5 points)



2. Assume that control is at the accept statement in line 16. What is the first word written on the screen after the student

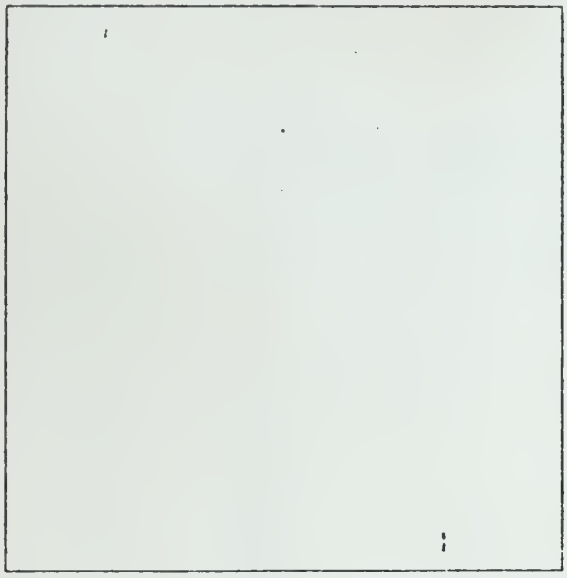
a. presses the back key? (3 points)

b. enters "1902"? (3 points)

c. enters "1899"? (3 points)

d. presses the help key? (3 points)

3. Assume that control is at the accept statement in line 16. In the box, reproduce (approximately) the contents of the screen at the next pause after the student types "1980" and then presses the help key. (8 points)

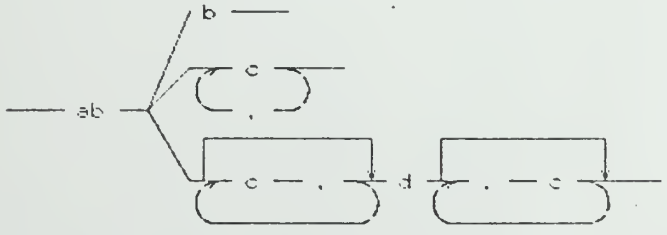


4. Assume control is at the pause statement in line 28. What is the first word written on the screen after the student

a. presses the help key? (3 points)

b. presses the next key? (3 points)

5.



The syntax diagram above is in the notation given in VAIDS. Which of the following are syntactically correct according to this diagram? (5 points)

a. ab

b. abc

c. abc,c

d. abcd

e. abc,c,d,c,c

B.3 Subject Grouping Schema

A function, randu, generated random choices from the set 1, 2. A 1 indicated that the first subject of a pair in the ranking would be assigned to group T, and a 2 indicated that the second subject would be assigned to group T.

SUBJECT GROUPING SCHEMA

X = randu(2)	Tk	KT
1	1	2
2	4	3
2	6	5
1	7	8
1	9	10
2	12	11
1	13	14
2	16	15
1	17	18
2	20	19
2	22	21
2	24	23
1	25	26
1	27	28
1	29	30
1	31	32
2	34	33
1	35	36
2	38	37
2	40	39

APPENDIX C

COMPLETE PROGRAM LISTINGS FOR LESSON "INTREP" IN T AND K

C.1 T Listing

```
1
2  lesson intrep;
3      integer nrsctns = 4,     ¢ number of sections ¢
4          nrqs = 9;            ¢ number of questions ¢
5      string(45) response;     ¢ current response ¢
6      integer ntries,          ¢ number of tries ¢
7          hipcnt,              ¢ hit count ¢
8          loc,                 ¢ location ¢
9          tnr,                 ¢ topic number ¢
10         qnr,                 ¢ question number ¢
11         qok ¢ false;         ¢ quiz ok now? ¢
12         i;
13     integer vector(4) tcompl ¢ false; ¢ topics complete ¢
14     integer vector(6) answers,       ¢ answers ¢
15         correctans(3288,1111B,2210,10000B1,10001,10100);
16                              ¢ correct answers ¢
17
18 procedure title;
19     i ¢ 1;
20     size 3;
21     *[ while i ¢ 53 :
22         at 172+i,398+i;  write INTEGER;
23         at 84+i,322+i;  write REPRESENTATION;
24         i ¢ i+1;
25     ]*;
26     size 8;
27     at 2229;  write Is it really true that  ;
28     size 1.5;  write ?;
29     size 6;  write 10 ;
30     size 1.5;  write ¢ 2;
31     size 2;  write 10 ;
32     size 1.5;  write ¢ 101;
33     size 2;  write (2);
34     write ?;
35     preavt:
36     end title;
37
38 procedure index;
39     back title;
40     term "index","contents";
41     erase;
42     at 322;  size 1.5;  write TABLE OF CONTENTS;  size 8;
43     loc ¢ 1137;
44     tnr ¢ 1;
45     *[ while i tnr≤nrsctns :
46         [ if i tcompl(tnr) :  at loc-2;  write *; ];
47         topics;
48         loc ¢ loc+322;
49         tnr ¢ tnr+1;
50     ]*;
51     loc ¢ [ if qok | 3013 | 30 ];
52     at loc;  write Press a, b, c, ;
53     [ if | qok :  write d, ; ];
54     write or NEXT.;
55     whereto_[ pause;  if key
56         | "a" :  jump intro;
57         | next :  jump intro;
58         | "b" :  jump mixedrad;
59         | "c" :  jump fibonac;
60         | "d" :
61             [ if qok | jump quiz; | at 3105;  write
62
63     You must complete every section before taking the quiz.;
64
65                 again whereto;
66             ];
67         | else : again;
68     ]_whereto;
69     end index;
70
71 procedure intro;
72     back intro;  back1 index;
73     at 321;  size 1.5;  write A. INTRODUCTION;  size 8;
74     at 905;  write
75
76 ~    An everyday, ordinary, base B integer N is represented
77 as
78
79         N = (b_n b_{n-1} ... b_2 b_1 b_0)_B
80
81           = b_n B^n + b_{n-1} B^{n-1} + ... + b_2 B^2 + b_1 B + b_0
82
83
84       Are there other ways to represent N?;
85
86     at 1915;  accept;
87     at 1915;  erase 2+length(reply);  at 1710;  erase 36;
88     at 1705;  write
89
90 ~    This is only one of many representations for N.  In
91 this lesson, (nrsctns-1) other systems are discussed:
92
93                       1. Mixed Radix
94
95                       2. Fibonacci;
96
97     tnr ¢ 1;
98     endtopic;
99     end intro;
100
101 procedure mixedrad;
```

```
    back mixedrad;  back1 index;  erase;
    at 311;
    size 1.5;  write MIXED RADIX REPRESENTATION;  size 0;
    at 605;  write

~   How many seconds in 1 day, 2 hours, 3 minutes, and
4 seconds?;

    ntries ⇐ 0;  hlpcnt ⇐ 0;
    inhibit erase;
    jump mixedr1;
    end mixedrad;

procedure mixedr1;
    back mixedrad;  back1 index;
    inhibit erase;
    help mrhelp;
    [ :if | hlpcnt>0 :  ¢ if necessary, reset screen ¢
        at 1818;  erase 29,3;
        at 1017;  erase 45;
        at 1317;  erase 31,2;
    ];
    [ at 1015;  long 45;  accept response;  long 150;
        ntries ⇐ ntries+1;
        put(",",","comma",response);
    if response
        | nearly '93 comma 784' :  ok;
        | nearly 93784 :  ok;
        | else :  no;  at 1317;
            [ case ntries
                | [ if opcnt(response)=0
                    | write Try using an Expression.;
                    | write Request help.;
                  ];
                | write Request help.;
                | write The answer is 93,784 or
                        1*24*60*60 + 2*60*60 + 3*60 + 4;
            ];
            retry;
    ];
    at 1325;  write

~   You used a mixed radix system to compute this result.
In this representation

        N = b_n a_n + b_{n-1} a_{n-1} + ... + b_1 a_1 + b_0 a_0

where a_0=x_0, a_1=x_1 x_0, ... a_n=x_n x_{n-1}...x_1 x_0 and the x_i's
are arbitrary positive integers.  In the system used in the
example above, a_0 = 1, a_1 = 60*1, a_2 = 60*60*1, and a_3 =
24*60*60*1.;

    wait;
    at 2405;  write

~   Notice that our everyday base 10 numbers are in mixed
radix notation with a_n = a^n.;

    at 3020;  write Press NEXT to continue.;
    end mixedr1;

procedure mixedr2;
    back mixedrad;  back1 index;
    erase;
    at 311;
    size 1.5;  write MIXED RADIX REPRESENTATION;  size 0;
    at 605;  write

~   Another useful mixed radix representation is the
factorial number system

                  i!
          a_i =
                  0  if i = 0;

    at 144,340;  size 5;  write {;  size 0;
    at 1310;  write

     The factorial representation is unique if 0 ≤ b_i ≤ i.;

    at 1610;  write Give the unique representation for 24.;
    ntries ⇐ 0;  hlpcnt ⇐ 0;
    inhibit erase;
    jump mixedr3;
    end mixedr2;

procedure mixedr3;
    back mixedrad;  back1 index;
    inhibit erase;
    loc ⇒ 1310;
    help mr2help;
    [ if | hlpcnt>0 :  ¢ if necessary, reset screen ¢
        at 2310;  erase 46,2;
        at 1817;  erase 45;
        at 2117;  erase 20;
    ];
    specs noops;
    [ at 1015;  long 45;  accept;  long 150;
        ntries ⇐ ntries+1;
    if reply
        | nearly 100000 :  ok;
            at 1815+length(reply)+7;
            [ case ntries
                | [illegible]
                | [illegible]
                | else :  write Good, you finally got it.;
```

```
            ];
        ! nearly 10*0 :  no;
            at 2117;  write What about b0?;
            retry;
        ! else :  at 2117;
            [ case ntries
                ! write 4' = 24;
                ! write Request help;
                ! write The answer is 10000.;
            ];
            retry;
    ];
    specs;
    at 2110;  write What decimal number equals 12010?;
    ntries = 2;  hlpcnt = 0;
    jump mixedr4;
    end mixedr3;

procedure mixedr4;
    back mixedr4;  back1 index;
    inhibit erase;
    loc = 2810;
    help mrthelp;
    [ if  hlpcnt>0 :  ¢ if necessary, reset screen ¢
        at 2810;  erase 46,2;
        at 2312;  erase 45;
        at 2617;  erase 17;
    ];
    [ at 1815;  long 45;  accept;  long 150;
        ntries = ntries+1;
    if reply
        ! nearly 37 :  ok;
            at 2315+length(reply)+7;
            [ case ntries
                ! write Excellent!;
                ! write Very good.;
                ! else :  write Good, you finally got it.;
            ];
        ! else :  at 2617;
            [ case ntries
                ! [ if cpcnt(reply)=0
                    ! write Try an expression;
                    ! write Request help;
                  ];
                ! write Request help;
                ! else :  write The answer is 37.;
            ];
            retry;
    ];
    tnr = 2;
    endtopic;
    end mixedr4;
```

```
procedure fibonac;
    back fibonac;  back1 index; erase;
    at 312;
    size 1.5;  write C. FIBONACCI REPRESENTATION.;  size 0;
    at 610;  write
        Do you remember how to calculate Fibonacci numbers?;
    at 815;  long 48;  accept;  long 150;
    at 815;  erase 50;  at 705;  write

 The following system defines a Fibonacci sequence.

                    F0 = 1
                    F1 = 2
                    Fi+1 = Fi + Fi-1;

    at 144,338;  size 5;  write {;  size 2;
    at 1405;  write

~    Give the first 5 numbers in this Fibonacci sequence.
(Separate input with commas or spaces.):

    ntries = 0;
    [ at 1815;  accept;  ntries = ntries+1;  if reply
        ! nearly '1 2 3 5 8' :  ok;
        ! else :
            [ case ntries
                ! write The first number (F0) is 1;
                        the second number (F1) is 2;
                        the nth number is Fn-1 + Fn-2.;
                ! write The first 3 numbers of the
                        sequence are 1,2,3.  The
                        fourth number is 2+3.;
                ! write The first 4 numbers of the
                        sequence are 1,2,3,5.  The
                        fifth number is 3+5.;
                ! else :  write The sequence is 1,2,3,5,8.;
            ];
        ];
    at 2005;  write

~    In the Fibonacci number system

          N = bnFn + bn-1Fn-1 + ... + b1F1 + b0F0.

     If 0 ≤ bi ≤ 1, and if no two adjacent bi's are 1, the
representation is unique.:

    at 3020;  write Press NEXT to continue.;
    end fibonac;

procedure fibonac2;
    back fibonac;  back1 index;
    at 312;
```

```
    size .5; write C. FIBONACCI REPRESENTATION; size 0;
    at 91 : write
        Give the Decimal representation for 1001.;
    ntries = 7; hlpcnt = 0;
    inhibit erase;
    jump fibonac3;
    end fibonac2;

procedure fibonac3;
    back fibonac; back1 index;
    inhibit erase;
    lcc = 1710;
    help fhelp;
    [ if hlpcnt>0 :  ¢ if necessary, reset screen ¢
        at 1710; erase 41,2;
        at 2217; erase 45;
        at 1517; erase 17;
    ];
    [ at 2215; long 45; accept; long 150;
        ntries = ntries+1;
    if reply
        [ 12 : ok;
            at 2215+length(reply)+7;
            [ case ntries
                ! write Excellent!;
                ! write Very good.;
                  write Good, you finally got it.;
            ];
        ! else : no;
            at 1517;
            [ if upcnt(reply)and(ntries<1)
                ! write Try an expression.;
                ! else :
                    [ case ntries
                        ! 52 : write Request help;
                        ! else : write The answer is 12.;
                    ];
            ];
    ];
    at 1925; write

                    Add: 2 + 2
     (Give the answer in Fibonacci Representation.);

    ntries = 0; hlpcnt = 0;
    jump fibonac4;
    end fibonac3;

procedure fibonac4;
    back fibonac; back1 index;
    inhibit erase;
    lcc = 2712;
    help fhelp;
    [ if hlpcnt>0 :  ¢ if necessary, reset screen ¢
        at 2717; erase 41,2;
        at 2217; erase 45;
        at 2517; erase 17;
    ];
    spcs nocps;
    [ at 2215; long 45; accept; long 150;
        ntries = ntries+1;
    if reply
        ! nearly 101 : ok;
            at 2215+length(reply)+7;
            [ case ntries
                ! write Excellent!;
                ! write Very good.;
                ! write Good, you finally got it.;
            ];
        ! nearly 4 : no; at 2517; write
            Fibonacci representation, please.;
        ! else : no;
            at 2517;
            [ case ntries
                ! write 2 5 3 ...;
                ! write Require help;
                ! else : write The answer is 23.;
            ];
    ];
    spcs;
    tnr = 3;
    endtopic;
    end fibonac4;

procedure quiz;
    inhibit term;
    inhibit erase;
    next1 report;
    at 329; size 1.5; write QUIZ; size 0;
    at 605; write

Give the answers in factorial (!) representation.


    1.  $17_{10} + 5_{10}$ =

    2.  $3010_{!} + 2100_{!}$ =

    3.  $15_{10} + 120_{!}$ =


Give answers in Fibonacci (F) representation.


    4.  $17_{10} + 5_{10}$ =
```

```
421
422       5.  1817₁₀ + 18₁₀  =
423
424       6.  15₈ + 10₂₀  =;
425
426      at 3007:  write
427          Press SHIFT-NEXT when you have comoleted the quiz.;
428      ansrs ← 0;  qnr ← 1;
429      *[ while ! qnr≤nrqs :  qstn;  qnr ← qnr+1; ]*;
430      *[ at 2805;  write
431              Which problem, if any, do you want to rework?;
432          [ qnr ← 0;
433              at where+2;  accept qnr;
434          if  ! (qnr<1)or(nrqs<qnr) :  again;
435          . ];
436          at 2805;  erase 59;
437          qstn;
438      ]*;
439      end quiz;
440
441  procedure prsnxt;
442      at 3027;  write Press NEXT;
443      end prsnxt;
444
445  procedure wait;
446      prsnxt;  pause next;  at 3027;  erase 10;
447      end wait;
448
449  procedure topics;
450      at 1cc;
451      [ case tnr
452          ! write a. Introduction;
453          ! write b. Mixed Radix Representation;
454          ! write c. Fibonacci Representation;
455          ! write d. Quiz;
456      ];
457      end topics;
458
459  procedure endtopic;
460      back1 :index;
461      tcompl(tnr) ← true;
462      at 2703;  write You have completed this topic:  ;
463      loc ← where;  topics;  write .;
464      at 2903;  write Press BACK if you wish to see it again.;
465      at 3023:
466      [ if tnr=(nrsctns-1)
467          ! write Press NEXT to return to the index.;
468          ! write Press NEXT for the next topic:  ;
469              loc ← where;  tnr ← tnr+1;  topics;  write .;
470      ];
471      at 3123:
472      write Press SHIFT-BACK for the table of contents.;
473      [ if ! qok :
474          qok ← true;
475          i ← 1;
476          *[ while ! i<nrsctns :
477              [ if ! tcompl(i)=false :  qok ← false; ];
478              i ← i+1;
479          ]*;
480      ];
481      [ if ! qok :  at 3027;  write
482          Press SHIFT-NEXT when you are ready for the quiz.;
483      ];
484      [ pause;  if key
485          ! next :  tnr ← tnr+1;
486              [ if ! tnr=nrsctns :  jump index; ];
487          ! next1 :  [ if ! qok :  jump quiz; ];
488              again:
489          ! back :
490          ! else :  again;
491      ];
492      [ case tnr
493          ! jump intro;
494          ! jump mixedrad;
495          ! jump fibonac;
496      ];
497      end endtopic;
498
499  procedure mrhelp;
500      inhibit erase;
501      hlpcnt ← hlpcnt+1;
502      at 1313;  write
503          There are 24 hours in a day,
504          60 minutes in an hour, and 60
505          seconds in a minute.;
506      endhelp mrhelp;
507
508  procedure mr2help;
509      inhibit erase;
510      hlpcnt ← hlpcnt+1;
511      at loc;
512      [ case hlpcnt
513          ! write N = b4a4 + b3a3 + b2a2 + b1a1 + b0a0;
514          ! write
515              N = b4(4!) + b3(3!) + b2(2!) + b1(1!) + b0(0!);
516          ! ≥ 3 : write
517              N = b4(24) + b3(6) + b2(2) + b1(1) + b0(0);
518      ];
519      endhelp mr2help;
520
521  procedure fhelp;
522      inhibit erase;
523      hlpcnt ← hlpcnt+1;
524      at loc;
525      [ case hlpcnt
526          ! write N = b4F4 + b3F3 + b2F2 + b1F1 + b0F0;
```

```
526              i ≥ 2 : write
527                  N = b4(8) + b3(5) + b2(3) + b1(2) + b0(1);
528          ];
529          enable file;
530
531      procedure getr;
532          loc ← 2200+qnr + [ if qnrs(nrqs div 2) | 731 | 1231 ];
533          at loc;  erase 38;
534          [ at loc; accept answr(qnr);  if | formerror :
535              at loc+length(reply)+7;  write Numbers only.;
536              pause rest;
537              at loc+length(reply)+7;  erase 13;
538              again;
539          ];
540          end getn;
541
542      procedure report;
543          inhibit term;
544          at 3207;  erase 57;
545          i ← 1;
546          *[ while i ≤nrqs :
547              loc ← 287+i + [ if i≤(nrqs div 2) | 704 | 1204 ];
548              at loc:  [ if corranswrs(i)=answrs(i) | ok; | no; ];
549          ]*;
550          at 3210;  write The lesson is complete.;
551          end report;
552
553      end intrep;
```

C.2 K Listing

```
1
2   lesson intrep;
3       integer nrsctns = 4,     ¢ number of sections ¢
4           nrqs = 5;            ¢ number of questions ¢
5       integer qok ← false;     ¢ quiz ok now? ¢
6       integer vector(4) tcompl ← false; ¢ topics complete ¢
7
8       ttl.        title;
9
10                  on "index","contents";
11      indx.       index;
12
13      intr.       intro;
14
15      mixdrad.    mixedrad;
16
17      fibon.      fibonac;
18
19      qz.         { disable term }quiz;
20
21  procedure title;
22      integer i;
23      erase;
24      i ← 1;
25      { size 3 }*[ while i ≤ 3 :
26          at 172+i,393+i;  write INTEGER;
27          at 84+i,382+i;   write REPRESENTATION;
28          i ← i+1;
29      ]*;
30      at 2209;  write Is it really true that  ;
31      { size 1.5 }write 2;
32      write ;
33      { size 1.5 }write + 2;
34      write ;
35      { size 1.5 }write = 101;
36      write (2);
37      write ?;
38      pause;
39      end title;
40
41  procedure index;
42      integer loc ¢location¢, tnr ¢topic number¢;
43      on back goto ttl;
44      erase;
45      at 320;  { size 1.5 }write TABLE OF CONTENTS;
46      loc ← 1117;
47      tnr ← 1;
48      *[ while i tnr≤nrsctns :
49          [ if i tcompl(tnr) :  at loc-2;  write ✓; ];
50          topics(loc,tnr);
51          loc ← loc+384;
52          tnr ← tnr+1;
53      ]*;
54      loc ← [ if qok : 3013 | 3013 ];
55      at loc;  write Press a, b, c, :
56      [ if i qok :  write d, : ];
57      write or RETURN.;
58      whereto_[ pause;  if key
59          i "a" :  goto intro;
60          i next :  goto intro;
61          i "b" :  goto mixedrad;
62          i "c" :  goto fibon;
63          i "d" :
64              [ if qok i goto qz; i at 3103;  write
65
66      You must complete every section before taking the quiz.;
67
68                  again whereto;
69              ];
70          i else : again;
71      ]_whereto;
72      end index;
73
74  procedure intro;
75      on back;  on back1 goto indx;
76      erase;
77      at 321;  { size 1.5 }write A. INTRODUCTION;
78      at 905;  write
79
80  ~   An everyday, ordinary, base B integer N is represented
81  as
82
83          N = (bnbn-1 ... b2b1b0)B:
84
85            = bnBn + bn-1Bn-1 + ... + b2B2 + b1B + b0
86
87
88      Are there other ways to represent N?:
89
90      at 1915;  accept;
91      at 1915;  erase 2+length(reply);  at 1718;  erase 36;
92      at 1705;  write
93
94  ~   This is only one of many representations for N.  In
95  this lesson, {nrsctns-1} other systems are discussed:
96
97                      1. Mixed Radix
98
99                      2. Fibonacci
100
101     endtopic(1);
```

```
    end intro;

procedure mixedrad;
    string(45) response;  ¢ student response ¢
    integer ntries,  ¢ number of tries ¢
        hlpcnt;  ¢ help count ¢
    on back:  on back: goto indx;
    erase;
    at 311:  { size 1.5 }write MIXED RADIX REPRESENTATION;
    at 585;  write

~   How many seconds in 1 day, 2 hours, 3 minutes, and
4 seconds?;

    [ on help nohelp;
        at 1215:  { long 45 }accept response;
        ntries ← ntries+1;
        put(",","comma",response);
    if response
        | nearly '93 comma 784' :  ok;
        | nearly 93784 :  ok;
        | else :  no;  at 1317;
            [ case ntries
                | [ if count(response)=0
                    | write Try using an Expression.;
                    | write Request help.;
                  ];
                | write Request help.;
                | write The answer is 93,784 or
                        1*24*60*60 + 2*60*60 + 3*60 + 4;
            ];
            retry;
    ];
    at 1325:  write

~   You used a mixed radix system to compute this result.
In this representation

        N = bnan + bn-1an-1 + ... + b1a1 + b0a0

where a0=x0, a1=x1x0, ... an=xnxn-1...x1x0 and the xi's
are arbitrary positive integers.  In the system used in the
example above, a0 = 1, a1 = 60*1, a2 = 60*60*1, and a3 =
24*60*60*1.;

    wait;
    at 2425:  write

~   Notice that our everyday base 10 numbers are in mixed
radix notation with an = a^n.;

    at 3020:  write Press NEXT to continue.;
    pause next;
    erase;
    at 311:  { size 1.5 }write MIXED RADIX REPRESENTATION;
    at 605:  write

~   Another useful mixed radix representation is the
factorial number system

                    i!
            ai =
                    0   if i = 0;

    at 144,340;  { size 5 }write {;
    at 1310:  write

     The factorial representation is unique if 0 < bi ≤ i.;

    at 1610:
    write Give the unique representation for 27.;
    ntries ← 0;
    hlpcnt ← 0;
    [ on help [ hlpcnt ← hlpcnt+1;  [illegible] ];
        at 1815:  { long 45 }accept;
        ntries ← ntries+1;
    if reply
        | nearly 10300 :  ok;
            at 18[illegible]+length[illegible]+7;
            [ case ntries
                | [illegible];
                | [illegible];
                | else :  write Good, you finally got it.;
            ];
        | nearly 1100 :  no;
            at 21[illegible]:  write What about b0?;
            retry;
        | else :  at 2117;
            [ case ntries
                | write 4! = 24;
                | write Request help;
                | write The answer is 1[illegible].;
            ];
            retry;
    ];
    at 2110:  write What decimal number equals [illegible]?;
    ntries ← 0;  hlpcnt ← 0;
    [ on help [illegible];  [illegible] ];
        at 1815:  { long 45 }accept;
        ntries ← ntries+1;
    if reply
        | nearly 27 :  ok;
            at 2315+length(reply)+7;
            [ case ntries
                | [illegible];
                | [illegible];
```

```
                ! else :  write Good, you finally got it.;
            ];
        ! else :  at 2517;
            [ case ntries
                ' [ if opent(reply)=0
                    ! write Try an expression;
                    ! write Request help;
                  ];
                ! write Request help;
                ! else :  write The answer is 37.;
            ];
            retry;
    ];
    endtopic(2);
    end mixedrad;

procedure fibonac;
    integer ntries hlpcnt;
    on back:  or back1 goto indx;
    erase;
    at 312;  [ size 1.5 ]write C. FIBONACCI REPRESENTATION;
    at 617;  write
        Do you remember how to calculate Fibonacci numbers?;
    at 815;  [ long 48 ]accept;  at 815;  erase 50;
    at 705;  write

The following system defines a Fibonacci sequence.

                    F0 = 1
                    F1 = 2
                    Fi+1 = Fi + Fi-1;

    at 144,338;  [ size 5 ]write f;
    at 1475;  write

~   Give the first 5 numbers in this Fibonacci sequence.
(Separate input with commas or spaces.);

    ntries = 0;
    [ at 1815;  accept;  ntries = ntries+1;  if reply
        ! nearly '1 2 3 5 8  :  ok;
        ! else :
            [ case ntries
                ! write The first number (F1) is 1;
                  the second number (F2) is 2;
                  the nth number is Fn-1 + Fn-2.;
                ! write The first 3 numbers of the
                  sequence are 1,2,3.  The
                  fourth number is 2+3.;
                ! write The first 4 numbers of the
                  sequence are 1,2,3,5.  The
                  fifth number is 3+5.;
                ! else :  write The sequence is 1,2,3,5,8.;
            ];
        ];
    at 2005;  write

~   In the Fibonacci number system

        N = bnFn + bn-1Fn-1 + ... + b1F1 + b0F0.

    If 0 ≤ bi ≤ 1, and if no two adjacent bi's are 1, the
representation is unique.;

    at 3020;  write Press NEXT to continue.;
    pause next;
    erase;
    at 510;  write
        Give the Decimal representation for 16? .;
    ntries = 0;  hlpcnt = 0;
    [ on help [ hlpcnt = hlpcnt+1;  fhelp(3510,hlpcnt); ];
        at 1215;  [ long 45 ]accept;
        ntries = ntries+1;
    if reply
        ! 12 :  ok;
            at 1215+length(reply)+7;
            [ case ntries
                ! write Excellent!;
                ! write Very good.;
                ! write Good, you finally got it.;
            ];
        ! else :  no;
            at 1517;
            [ if opent(reply)=0 or ntries<4;
                ! write Try an expression;
                ! else :
                    [ case ntries
                        ! 62 :  write Request help;
                        ! else :  write The answer is 12.;
                    ];
            ];
    ];
    at 1895;  write

~                   Q4: 2 + 3
    (Give the answer in Fibonacci Representation.);

    ntries = 0;
    hlpcnt = 0;
    [ on help [ hlpcnt = hlpcnt+1;  fhelp(3510,hlpcnt); ];
        at 2215;  [ long 45 ]accept;
        ntries = ntries+1;
    if reply
        ! nearly 101 :  ok;
            at 2215+length(reply)+7;
```

```
                [ case ntries
                    ! write Excellent!;
                      ! write Very good.;
                    ! write Good, you finally got it.;
                ];
            ! really 4 : no:  at 2517:  write
                Fibonacci representation, please.;
            ! else :  no:
                at 2517;
                [ case ntries
                    ! write 3 5 3 2 1;
                    ! write Request help;
                    ! else :  write The answer is 101.;
                ];
        ];
        endtopic(3);
        end fibonacci;

    procedure quiz;
        integer loc,        ¢ location ¢
            qnr,            ¢ question number ¢
            i;
        integer vector(nrqs) answrs:=0,     ¢ answers ¢
            corranswrs:=(3200,11110,2210,1000001,10001,10100);
                            ¢ correct answers ¢
        erase;
        at 209;  [ size 1.5 ]write QUIZ;
        at 695;  write

Give the answers in factorial (!) representation.


        1.  $17_{10} + 5_{10}$  =

        2.  $7012_{!} + 2100_{!}$  =

        3.  $15_{10} + 100_{!}$  =


Give answers in Fibonacci (F) representation.


        4.  $17_{10} + 5_{10}$  =

        5.  $1012_{F} + 12_{F}$  =

        6.  $15_{10} + 100_{F}$  =;

        at 3007;  write
            Press SHIFT-NEXT when you have completed the quiz.;
        answrs ¢ 0;  qnr ¢ 1;
        *[ while [ qnr≤nrqs : qstn(qnr,next1);  qnr ¢ qnr+1; ]*;
        *[ at 2005;  write
                Which problem, if any, do you want to review?;
            [ qnr ¢ 1:
                at 2025;  accept qnr;
            ii  [ (qnr<1)or(nrqs<qnr)  :  again;
            ];
            at 2025;  erase 73;
            qstn(qnr,next1);
        ]*;
        on next1;
        at 3007;  erase 57;
        i ¢ 1;
        *[ while [ i≤nrqs :
            loc ¢ 200*i + [ if i≤nrqs div 2 [ 784 - 124*i ];
            at loc:  [ if corranswrs(i)=answrs(i) [ ok; [ no; ];
        ]*;
        at 3000;  write The lesson is complete.;
        end quiz;

    procedure prsnxt;
        at 3027;  write Press NEXT;  pause next;
        end prsnxt;

    procedure wait;
        prsnxt;  pause funct;  at 3027;  erase 18;
        end wait;

    procedure topics( integer loc, nbr );
        at loc;
        [ case nbr
            ! write a. Introduction;
            ! write b. Mixed Radix Representation;
            ! write c. Fibonacci Representation;
            ! write d. Quiz;
        ];
        end topics;

    procedure exittopic( integer tnbr );
        integer i;
        on back1 goto ind;
        tcompl(tnbr) ¢ true;
        at 2805;  write You have completed this topic;
        topics(where,tnbr);  write ;
        at 2903;  write Press BACK if you wish to see it again.;
        at 3003;
        [ if tnbr<(nrsctns-1)
            ! write Press NEXT to return to the index.;
            ! write Press NEXT for the next topic:
                topics(where,tnbr+1 :  write ;
        ];
        at 3103;
        write Press SHIFT-BACK for the table of contents.;
        i ¢ 1;
```

```
420      *[ while i <=nmsctns :
421          [ i* toomch(i)=false :  signal incompl; ];
422          i := i+1;
423      ]*;
424      qok := true;
425      at 3272;
426      write Press SHIFT-NEXT when you are ready for the quiz.;
427      on incompl:
428      [ pause;  if key
429          | next :  tnr := tnr+1;
430              [ if i tnr>nmsctns :  goto indx; ];
431          | next1 :  [ if i qok :  goto qz; ];
432              again;
433          | back :
434          | else :  again;
435      ];
436      [ case tnr
437          | goto intr;
438          | goto mixdrad;
439          | goto fibon;
440      ];
441      end enotecic;
442
443  procedure mrhelp;
444      at 1812:  write
445          There are 24 hours in a day,
446          60 minutes in an hour, and 60
447          seconds in a minute.;
448      pause next;
449      at 1812:  erase 29,3;
450      at 1317;  erase 45;  at 1317;  erase 31,2;
451          ¢ reset screen ¢
452      end mrhelp;
453
454  procedure mr2help( integer loc,hipcnt );
455      at loc;
456      [ case hipcnt
457          | write N = b4a4 + b3a3 + b2a2 + b1a1 + b0a0;
458          | write
459              N = b4(4!) + b3(3!) + b2(2!) + b1(1!) + b0(0!);
460          | else write
461              N = b4(24) + b3(6) + b2(2) + b1(1) + b0(0);
462      ];
463      pause next;
464      at loc:  erase 46,2;
465      at loc-500+7;  erase 45;  at loc-200+7;  erase 20;
466          ¢ reset screen ¢
467      end mr2help;
468
469  procedure fhelp( integer loc,hipcnt );
470      at loc;
471      [ case hipcnt
472          | write N = b4F4 + b3F3 + b2F2 + b1F1 + b0F0;
473          | else write
474              N = b4(0) + b3(5) + b2(3) + b1(2) + b0(1);
475      ];
476      pause next;
477      at loc:  erase 41,2;
478      at loc-500+7;  erase 45;  at loc-200+7;  erase 13;
479          ¢ reset screen ¢
480      end fhelp;
481
482  procedure qstn( integer qnr; on next1 ):
483      integer loc;
484      loc := 200*qnr + [ if qnrs(nrqs div 2)  777 | 1231 ];
485      at loc;  erase 30;
486      on formerror [
487          at loc+length(reply)+7;  write Numbers only.;
488          pause next;
489          at loc+length(reply)+7;  erase 13;
490      ];
491      at loc:  accept ansrs(qnr);
492      end qstn;
493
494  end intrep;
```

APPENDIX D

PERFORMANCE STATISTICS FOR THE DEBUGGING SECTION

D.1 Scores

Bug	Means		df	T	p(two-tailed)
	K	T	26		
1	5.8	5.5		.19	
2	6.5	6.1		.25	
3	5.2	7.6		2.16	.040
4	4.9	7.2		1.39	
5	4.2	4.9		.37	
6	.8	2.3		1.60	
7	.4	1.8		1.27	
8	.1	.2		.72	
Total	27.8	35.7		1.18	

Table D1. Scores: Includes All Subjects (13 K, 15 T)

Bug	Number of Subjects		Means		df	T	p(two-tailed)
	K	T	K	T			
1	13	15	5.8	5.5	26	.19	
2	13	14	6.5	6.6	25	.02	
3	12	14	5.6	8.1	24	2.69	.013
4	10	13	6.4	8.3	21	1.27	
5	8	11	6.9	6.6	17	.13	

Table D2. Scores: Includes Only Subjects Who Worked the Problem

Bug	Number of Subjects		Means		df	T	p(two-tailed)
	K	T	K	T			
1	13	15	6.8	6.7	26	.10	
2	12	14	7.5	7.6	24	.07	
3	10	14	8.6	11.2	22	1.52	
4	8	11	11.3	13.7	17	1.21	
5	5	8	14.0	9.6	11	2.22	.049

Table D3. Scores Adjusted to Give More Credit to Subjects Who Performed Well Quickly

D.2 Timing Data

The timing data consisted of the following variables:

1. time spent executing "intrep"
2. time spent looking at the listing before moving forward
3. time spent looking at the listing before searching for text
4. time spent looking at the listing before moving backward
5. time spent inserting
6. time spent replacing
7. time spent looking at the listing before deleting
8. time spent on miscellaneous actions including time spent looking at the listing before inserting and replacing and before accessing the on-line aids
9. time spent reading the reference manual
10. time spent looking at the flow diagram
11. time spent obtaining help on the use of the editor
12. sum of 2, 3, 4, and 8: time spent looking at the listing
13. sum of 5, 6, and 7: time spent altering the listing
14. sum of 9, 10, and 11: time spent referencing the on-line aids
15. total time spent

Variable	Means		df	T	p(two-tailed)
	K	T	26		
1	114.8	119.9		.10	
2	77.2	66.3		.42	
3	29.0	11.6		1.86	.074
4	9.9	17.5		.87	
5	6.8	25.3		1.26	
6	94.2	28.3		2.89	.008
7	.0	3.5			
8	71.2	80.3		.36	
9	11.1	11.6		.05	
10	.8	.0			
11	18.0	15.2		.24	
12	187.4	175.7		.23	
13	101.1	57.1		1.70	
14	29.8	26.8		.17	
15	433.2	379.5		.52	

Table D4. Timing Data for Bug 1 (13 K, 15 T)

Variable	Means		df	T	p(two-tailed)
	K	T	24		
1	115.4	109.6		.38	
2	125.7	105.9		.52	
3	70.3	61.0		.42	
4	21.1	31.1		.88	
5	.0	18.7			
6	.6	3.4		1.28	
7	7.5	.0			
8	88.8	70.9		.90	
9	5.8	7.8		.28	
10	3.1	.8		1.06	
11	7.6	4.6		.70	
12	305.8	268.9		.75	
13	8.3	22.1		2.83	.009
14	16.5	13.1		.33	
15	446.0	413.8		.53	

Table D5. Timing Data for Bug 2 (12 K, 14 T)

Variable	Means		df	T	p(two-tailed)
	K	T	22		
1	44.9	45.7		.07	
2	31.3	27.4		.33	
3	6.7	6.8		.01	
4	13.6	25.0		1.07	
5	39.4	22.6		2.33	.030
6	4.9	9.6		.76	
7	.0	.4			
8	42.0	35.0		.85	
9	2.7	14.6		1.23	
10	.0	.0			
11	1.1	.0			
12	93.6	94.2		.02	
13	44.3	32.6		1.13	
14	3.8	14.6		1.11	
15	186.6	187.1		.01	

Table D6. Timing Data for Bug 3 (10 K, 14 T)

Variable	Means		df	T	p(two-tailed)
	K	T	17		
1	36.8	30.7		.70	
2	30.1	35.1		.48	
3	13.9	7.0		.78	
4	6.9	.8		1.89	.080
5	16.9	11.1		1.65	
6	3.5	2.9		.19	
7	3.0	.0			
8	42.4	27.8		1.39	
9	.0	1.5		.85	
10	.0	.6			
11	.0	.0			
12	93.3	70.7		1.34	
13	23.4	14.0		1.66	
14	.0	2.1			
15	153.4	117.5		1.44	

Table D7. Timing Data for Bug 4 (8 K, 11 T)

Variable	Means		df	T	p(two-tailed)
	K	T	11		
1	55.0	57.9		.23	
2	32.4	66.4		2.87	.015
3	12.0	7.4		.71	
4	9.0	12.8		.32	
5	9.4	10.3		.15	
6	2.4	2.6		.08	
7	.0	3.5			
8	23.8	30.0		.64	
9	.0	.5			
10	.0	.0			
11	.0	1.3			
12	77.2	116.5		1.92	.081
13	11.8	16.4		.63	
14	.0	1.8			
15	144.0	192.0		1.51	

Table D8. Timing Data for Bug 5 (5 K, 8 T)

Variable	Means		df	T	p(two-tailed)
	K	T	22		
1	355.1	337.8		.50	
2	326.9	313.9		.25	
3	136.5	117.8		.46	
4	48.8	102.7		3.43	.002
5	55.9	86.2		1.89	.072
6	109.4	58.8		2.03	.056
7	9.1	5.7		.74	
8	253.1	259.1		.34	
9	31.2	38.6		.34	
10	3.9	1.5		.99	
11	30.3	18.8		.77	
12	765.3	739.5		.60	
13	174.4	150.7		.94	
14	65.3	58.8		.26	
15	1360.2	1340.8		1.46	

Table D9. Timing Data Totals for All Subjects Who Completed the Debugging Section

D.3 Editing Data

The editing data consisted of the following variables:

1. number of executions of "intrep"
2. number of forwards, F
3. number of searches, X
4. number of backs, B
5. number of inserts, I
6. number of replaces, R
7. number of deletes, D
8. number of editing keys incorrectly entered or partially entered and then erased
9. number of accesses to the reference manual
10. number of references to the flow diagram
11. number of requests for help in editing
12. sum of 2, 3, and 4: number of searching options activated
13. sum of 5, 6, and 7: number of alterations
14. sum of 9, 10, and 11: number of references to the on-line aids
15. sum of 1, 12, 13, and 14: total editing keypresses

Variable	Means		df	T	p(two-tailed)
	K	T	26		
1	1.6	1.9		.87	
2	4.4	4.7		.19	
3	1.9	.8		1.64	
4	.8	1.4		1.03	
5	.2	.7		1.63	
6	1.1	.9		.43	
7	.0	.4			
8	2.9	3.2		.29	
9	.2	.2		.30	
10	.8	.0			
11	.8	.7		.23	
12	7.0	6.9		.03	
13	1.2	2.1		1.39	
14	1.0	.9		.27	
15	10.8	11.8		.29	

Table D10. Editing Data for Bug 1 (13 K, 15 T)

Variable	Means		df	T	p(two-tailed)
	K	T	24		
1	1.5	2.1		1.40	
2	8.7	7.5		.36	
3	5.1	4.1		.70	
4	1.6	2.4		.89	
5	.0	.9			
6	.1	.2		.90	
7	.5	.0			
8	4.7	3.1		1.41	
9	.2	.1		.16	
10	.3	.1		.98	
11	.7	.4		.79	
12	15.3	14.0		.35	
13	.7	1.1		1.84	.078
14	1.1	.6		1.00	
15	18.6	17.8		.19	

Table D11. Editing Data for Bug 2 (12 K, 14 T)

Variable	Means		df	T	p(two-tailed)
	K	T	22		
1	1.3	1.4		.23	
2	1.8	2.1		.46	
3	.7	.9		.22	
4	1.0	2.1		1.27	
5	1.1	1.2		.56	
6	.2	.4		.98	
7	.0	.1			
8	1.9	1.8		.31	
9	.1	.2		.72	
10	.0	.0			
11	.2	.0			
12	3.5	5.1		.90	
13	1.3	1.7		1.21	
14	.3	.2		.39	
15	6.4	8.4		.96	

Table D12. Editing Data for Bug 3 (10 K, 14 T)

Variable	Means		df	T	p(two-tailed)
	K	T	17		
1	1.3	1.2		.34	
2	2.0	2.2		.38	
3	1.1	.4		1.46	
4	.5	.1		1.64	
5	1.3	.8		2.16	.045
6	.3	.2		.34	
7	.2	.0			
8	1.6	1.9			
9	.0	.1			
10	.0	.1			
11	.0	.0			
12	3.6	2.6		2.71	.015
13	1.8	1.0			
14	.0	.2			
15	6.6	5.0		2.23	.040

Table D13. Editing Data for Bug 4 (8 K, 11 T)

Variable	Means		df	T	p(two-tailed)
	K	T	11		
1	1.2	1.5		.67	
2	3.6	4.9		1.14	
3	1.4	.9		.72	
4	.6	1.3		.63	
5	1.0	1.0			
6	.2	.3		.19	
7	.0	.5			
8	1.2	2.5			
9	.0	.1			
10	.0	.0			
11	.0	.0			
12	5.6	7.1		.84	
13	1.2	1.7		.68	
14	.0	.3			
15	8.0	10.6		1.08	

Table D14. Editing Data for Bug 5 (5 K, 8 T)

Variable	Means		df	T	p(two-tailed)
	K	T	22		
1	6.7	9.2		3.38	.003
2	22.3	23.2		.18	
3	9.6	9.4		.05	
4	4.1	8.3		3.06	.006
5	2.4	4.5		2.94	.007
6	2.8	2.6		.19	
7	.7	.7		.00	
8	11.2	13.2		1.27	
9	.6	.9		.73	
10	.3	.2		.76	
11	1.8	1.1		.98	
12	36.0	40.9		1.05	
13	5.8	7.8		1.53	
14	2.6	2.5		.58	
15	51.2	60.0		1.73	.098

Table D15. Editing Data Totals for All Subjects Who Completed the Debugging Section

APPENDIX E

PERFORMANCE STATISTICS FOR THE MODIFYING SECTION

E.1 Scores

The modifying section of the experiment was scored as follows:

1. Specification #1
    a. Control Linkage (15 points)
    b. Procedure (25 points)
2. Specification #2
    a. Control Linkage (35 points)
    b. Procedure (25 points)

Two K subjects completed specification 1, and then inadvertently exited the experiment.

Item Scored	Means		df	T	p(two-tailed)
	K	T			
Control Linkage 1	11.9	14.1	23	1.34	
Procedure 1	16.8	18.8	23	.97	
Specification 1	28.7	32.8	23	1.46	
Control Linkage 2	9.1	8.8	21	.13	
Procedure 2	19.4	15.2	21	1.96	.063
Specification 2	28.5	24.1	21	1.30	
Total Performance	56.0	56.9	21	.16	

Table E1. Scores for the Modifying Section

E.2 Timing Data

The timing data consisted of the following variables:

1. time spent reading the specifications initially
2. subsequent time spent reading the specifications
3. time spent looking at the listing before moving forward
4. time spent looking at the listing before searching for text
5. time spent looking at the listing before moving backward
6. time spent inserting
7. time spent replacing
8. time spent looking at the listing before deleting
9. time spent on miscellaneous actions including time spent looking at the listing before inserting and replacing and before accessing the on-line aids
10. time spent reading the reference manual
11. time spent looking at the flow diagram
12. time spent obtaining help on the use of the editor
13. sum of 1 and 2: time spent reading the specifications
14. sum of 3, 4, 5, and 9: time spent looking at the listing
15. sum of 6, 7, and 8: time spent altering the listing
16. sum of 10, 11, and 12: time spent referencing the on-line aids
17. total time spent excluding the time spent reading the specifications initially

Variable	Means		df	T	p(two-tailed)
	K	T	21		
1	100.0	106.8		.51	
2	135.0	104.9		1.03	
3	151.5	148.5		.06	
4	68.0	124.2		1.49	
5	97.9	108.2		.30	
6	423.6	379.8		.93	
7	107.3	47.3		2.58	.017
8	6.4	12.8		1.23	
9	283.7	271.8		.34	
10	48.0	98.8		1.30	
11	1.7	4.3		1.10	
12	11.4	5.0		1.31	
13	235.0	211.8		.72	
14	601.1	652.6		.75	
- 15	537.3	439.8		2.01	.058
16	61.1	108.2		1.16	
17	1434.5	1412.4		.28	

Table E2. Timing Data for the Modifying Section

E.3 Editing Data

The editing data consisted of the following variables:

1. number of references to the specifications
2. number of forwards, F
3. number of searches, X
4. number of backs, B
5. number of inserts, I
6. number of replaces, R
7. number of delets, D
8. number of editing keys incorrectly entered or partially entered and then erased
9. number of accesses to the reference manual
10. number of references to the flow diagram
11. number of requests for help in editing
12. sum of 2, 3, and 4: number of searching options activated
13. sum of 5, 6, and 7: number of alterations
14. sum of 9, 10, and 11: number of references to the on-line aids
15. sum of 1, 12, 13, and 14: total editing keypresses

Variable	Means		df	T	p(two-tailed)
	K	T	21		
1	3.6	3.5		.21	
2	10.5	12.2		.37	
3	5.2	11.1		1.75	.095
4	9.2	10.0		.32	
5	6.9	9.3		2.65	.015
6	3.3	2.7		.45	
7	.7	1.5		1.25	
8	9.1	10.1		.64	
9	.8	1.8		1.87	.076
10	.1	.4		1.55	
11	.9	.4		1.40	
12	24.9	33.2		1.58	
13	10.9	13.7		1.93	.068
14	1.8	2.5		1.03	
15	41.2	52.9		2.10	.048

Table E3. Editing Data for the Modifying Section

APPENDIX F

CORRELATIONS

A missing data correlation program was used to compute these statistics.

Correlation factors:

1. University status
2. Courses
3. Languages
4. PLATO experience
5. KAIL quiz score
6. Rank
7. Score on debugging section
8. Time spent in "intrep"
9. Time spent looking at listing (debugging)
10. Time spent altering listing (debugging)
11. Total editing keypresses (debugging)
12. Score on modifying section
13. Time spent reading specifications
14. Time spent looking at listing (modifying)
15. Time spent altering lsiting (modifying)
16. Total editing keypresses (modifying)

For correlation factors 1 through 6, 13 K and 15 T subjects were considered; for correlation factors 7 through 11, 12 K and 12 T subjects were considered; and for correlation factors 12 through 16, 10 K subjects and 13 T subjects were considered. In the following tables

* = significant with probability .05, and

** = significant with probability .01.

F.1 Correlations for T Language Subjects

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1																
2	.482*															
3	.086	.506*														
4	-.157	-.182	-.091													
5	.086	.236	.366	.271												
6	-.298	-.502*	-.553*	-.301	-.906**											
7	-.313	.131	.379	.239	.442	-.567*										
8	.266	-.161	-.455	.286	.076	.037	.012									
9	-.468	-.243	-.050	.164	-.512*	.482*	-.438	-.115								
10	.129	.045	-.063	-.301	-.100	.076	-.037	-.707**	-.207							
11	.074	.028	-.383	.108	-.348	.352	-.541*	.532*	.453	-.438						
12	.178	.105	.210	.507*	.152	-.309	.645**	.435	.035	-.616*	.042					
13	.189	.439	.147	-.677**	.012	.030	-.393	.038	-.034	-.128	.350	-.402				
14	-.235	-.088	-.209	-.038	.077	.158	-.546*	.451	.264	-.314	.493*	-.269	.343			
15	.097	-.116	-.210	.096	-.420	.351	-.172	-.094	.263	.132	.431	.024	.004	-.426		
16	-.295	.094	.079	-.159	.124	-.021	-.399	.158	.171	-.285	.583*	.487*	.143	.656**	-.536*	

Table F1. T Correlations

F.2 Correlations for K Language Subjects

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1																
2	.506*															
3	-.016	.464*														
4	-.534*	.154	.637**													
5	.044	.421	.368	.382												
6	-.183	-.729**	-.588*	-.544*	-.866**											
7	.000	.542*	.735**	.785**	.717**	-.901**										
8	-.154	.055	-.360	-.302	-.517*	.434	-.500*									
9	-.199	-.034	.425	.461	.492*	-.435	.482*	-.814**								
10	-.148	.073	-.114	-.347	-.568*	.387	-.381	.553*	-.742**							
11	-.210	-.459	-.217	-.244	-.751**	.747**	-.587*	.249	-.348	.633*						
12	.078	.548*	.779**	.644*	.601*	-.871**	.905**	-.422	.462	-.486	.497					
13	.094	-.278	-.607	-.551*	-.163	.399	-.654*	.147	-.160	-.016	-.104	-.412				
14	-.222	.402	.397	.355	.242	-.415	.386	-.225	.668*	-.484	.267	.293	-.156			
15	.238	-.325	-.543*	-.615*	-.293	.455	-.473	.079	-.022	-.058	-.131	-.410	.134	-.039		
16	-.623*	.308	.159	.366	.286	-.293	.154	.406	.016	.027	.139	.018	.002	.505	-.422	

Table F2. K. Correlations

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-76-771	2.	3. Recipient's Accession No.

4. Title and Subtitle

AN EXPERIMENT ON CAI SEQUENCING CONSTRUCTS

5. Report Date: February 1976

6.

7. Author(s): David W. Embley

8. Performing Organization Rept. No. UIUCDCS-R-76-771

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

10. Project/Task/Work Unit No.

11. Contract/Grant No. US-NSF-EC-41511

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, D.C.

13. Type of Report & Period Covered: Technical Report

14.

15. Supplementary Notes

16. Abstracts

This report describes and gives the results of an experiment designed to investigate sequencing constructs in computer-aided instruction (CAI) programs. The experiment compared two languages. One incorporated sequencing constructs of an established CAI programming language; the other contained alternate constructs which stressed uniformity, orthogonality, and static definition and minimized the use of hidden side effects.

Working on-line, subjects debugged and modified a large program while the system monitored their progress and gathered data. Although several interesting results emerged, it was impossible to state conclusively that the entire set of sequencing constructs in either language proved better. Instead, the experiment indicated that direct, explicit, and simple constructs are best. Recommendations are given for designing sequencing features for CAI languages.

17. Key Words and Document Analysis. 17a. Descriptors

Programming Languages
Programming Language Design
Psychological Experiments in Programming
Computer-Assisted Instruction

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release Unlimited	UNCLASSIFIED	106
	20. Security Class (This Page) UNCLASSIFIED	22. Price